Datapath Synthesis for a 16-bit Microprocessor

Haobo Yu and Daniel Gajski

CECS Technical Report 02-05
January 22, 2002

Center for Embedded Computer Systems
Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8059

{haoboy,gajski}@ics.uci.edu

Datapath Synthesis for a 16-bit Microprocessor

Haobo Yu and Daniel Gajski

CECS Technical Report 02-05
January 22, 2002

Center for Embedded Computer Systems
Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425,USA
(949) 824-8059

{haoboy,gajski}@ics.uci.edu

Abstract

In this report, we'll describe the datapath synthesis for a simple 16-bit microprocessor using our own RTL synthesis tool. The initial part of this report introduces the instruction set of the processor as well as its instruction set super FSMD model. Then we further develop into different implementations of the processor's datapath. We will try different resource allocation combinations to the design and perform the synthesis on different target RTL structure with our tool. We then analyze the performance of these implementations on the basis of synthesis results from our tool and show how the designer has the choice to make the ultimate decision about the design with due considerations to all involved tradeoffs.

Contents

List of Figures

Datapath Synthesis for a 16-bit Microprocessor

Haobo Yu and Daniel Gajski
Center for Embedded Computer Systems
Information and Computer Science
University of California, Irvine

Abstract

In this report, we'll describe the datapath synthesis for a simple 16-bit microprocessor using our own RTL synthesis tool. The initial part of this report introduces the instruction set of the processor as well as its instruction set super FSMD model. Then we further develop into different implementations of the processor's datapath. We will try different resource allocation combinations to the design and perform the synthesis on different target RTL structure with our tool. We then analyze the performance of these implementations on the basis of synthesis results from our tool and show how the designer has the choice to make the ultimate decision about the design with due considerations to all involved tradeoffs.

1. Introduction

With the ever increasing complexity and time-to-market pressures in the design of embedded systems, designers have moved the design to higher levels of abstraction in order to increase productivity. However, each design must be described, eventually, at the lower level(e.g. layout masks) through various refinement processes. High-level synthesis has been recognized as one of the major design refinement processes.

The high-level synthesis involves the transformation of behavioral description of the design into a set of interconnected register transfer components which satisfy the behavior and some specified constraints, such as the number of resources, timing and so on. Three major synthesis tasks are applied during the transformation: allocation, scheduling, and binding. Allocation determines the number of the resources, such as storage units, buses, and function units, that will be used in the implementation. Scheduling partitions the behavioral description into time intervals. Binding assigns variables to storage units(storage binding), assigns operations to function units(function binding), and interconnections to buses(connection binding).

Many researches for High-level synthesis [GDLW92] have been done since 1980s. Currently, many commercial and academical high-level synthesis tools exist in electronic design automation market but the design community wouldn't integrate them into its design methodology and design flow by the following reasons:

- they can support only several limited architectures like multiplexer-based architecture
- they lack interaction between tools and the designers
- the quality of the generated design is worse than that of mannual design.

To make them popularly used in design community, we should tackle these problems. We propose a RTL design methodogy, which is based on Accellera RTL semantics proposed by Accellera C/C++ Working Group [Acc01]. Our target architecture for the RTL design methodology is bus-based architecture instead of mux-based architecture in which all RTL components such as function units and storage units are connected through buses to transfer data. because the performance of bus-based architecture is better than that of mux-based architecture in large design [Acc01]. Also the function/storage units are pipelined or multi-cycled in our target architecture. The storage units can be composed of registers, register files and memories with different latency and pipeline scheme. In other word, target architecture is heterogenous in terms of storage units. The RT components are connected through the allocated buses from ports of function units and storage units.

In this paper, we will demonstrate how our RTL synthesis tool works by synthesizing the datatpath of a 16-bit microprocessor.We will see how our RTL synthesis tool can be exploited to generate different datapath for the microprocessor.

The rest of this report is organized as follows: Section 2 gives an insight into how our RTL synthesis tool works. Section 3 describes the instruction set for the microprocessor as well as its instruction set super FSMD models. In section 4 we compare and analyze the experimental results after performing the synthesis on different implementations of the processor using our RTL synthesis tool. Section 5 concludes this report with a brief summary and future works.

2. Datapath Synthesis

Our tool synthesizes a design from a RTL behavior description in style 1 to style4 [ZSY$^+$00]. This tool performs four different tasks: scheduling, storage unit binding, functional unit binding and bus binding. The scheduling takes place first followed by the different binding. Here we use resource constraint binding algorithms in which the type and the number of of resources to be used are specified by the designer. The designer can let the tool synthesize different implementations with varying resource allocation combinations. The central idea is that after a designer specifies the resource combination to be used in in the target architecture, such as register files, functional units and buses, the tool synthesizes the design into an implementation that makes complete utilization of these allocated resources and at the same time minimize the cost of the interconnections, i.e. minimize the number of multiplexors and bus drivers.

2.1 RTL structure exploration flow

Most high level synthesis tools are built to do everything automatically. Research is focused on how to minimize the number of operation units, resources storage units and interconnection units (multiplexors and number of connections). Nearly all the synthesis tools are trying to explore the design space automatically without human intervention. But all these automatic approaches, though good in intention, failed to achieve satisfactory synthesis quality. The automatic tools can't explore such broad design space by themselves. We need the designer to participate in the design space exploration process, because the designer has more specific knowledge and experience about the direction of exploration. By using our tool,the user can compare the performance of different implementations according to the synthesis result and finds the best implementation with due consideration to the cost-performance tradeoff.

Figure 1 shows the flow of our designer directed design space exploration approach. First, the user specifies the target architecture and allocates the corresponding resource according to the target architecture, then our synthesis tool does scheduling/binding based on the specified resources and produces cycle accurate FSMD code. The output code is similar to the instruction set super FSMD except for the fact that some super FSMD states have been broken into several clock cycles to eliminate data dependencies and satisfy resource constraints. If our tool fails to produce the synthesis result, the designer allocates more resources, this interaction is repeated until the tool can produce the required

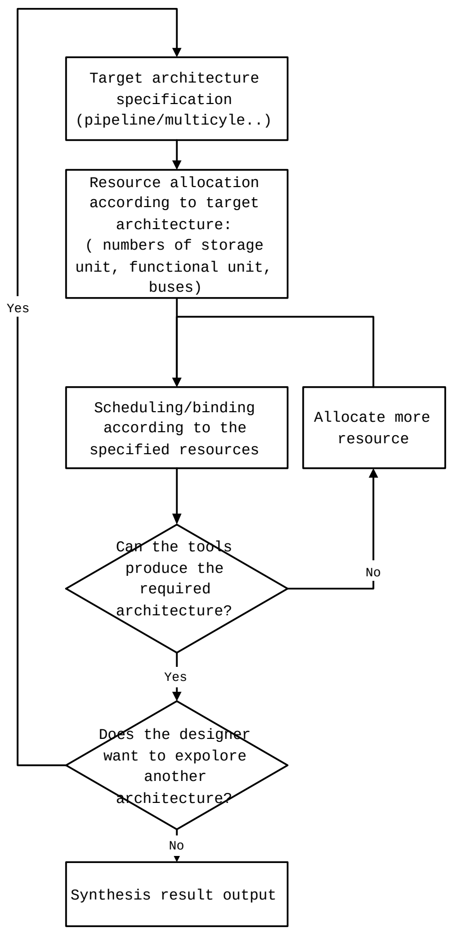

Target architecture specification (pipeline/multicyle..)
Resource allocation according to target architecture: (numbers of storage unit, functional unit, buses)
Scheduling/binding according to the specified resources
Allocate more resource
Can the tools produce the required architecture?
No
Yes
Does the designer want to expolore another architecture?
Yes
No
Synthesis result output

Figure 1. RTL structure exploration flow

architecture. Then, the designer can try another target architecture and the whole process is repeated again, by this way, we give the designer more freedom to explore the design space. Since the experienced designer has much knowledge about the design, his feedback and direction in this interactive exploration process will lead to better synthesis result than the automatic procedure.

3. Instruction Set Description

A 16-bit microprocessor [Gaj97] can access 64K of memory with one word of data. To reduce the number of memory accesses during the instruction fetch, we limit the instruction size to at most two memory words, which means that we can only use one-address instructions when accessing memory. Therefore, each instruction would consist of one or two 16-bit words: the second word, if used, would be a memory address, while the first word would specify the instruction type, the operation code and the register file addresses. In order to accommodate three register file addresses, we have to divide the 16-bit instruction into five fields: the `Type` field (2-bits), the `Op` field (5-bits), and three register file addresses identified as `Dest` (3-bits), `Src1` (3-bits) and `Src2`(3-bits). Examples of instructions from the instruction set are shown in Figure 2.

The instruction set includes four different types, *register* , *memory*, *control* and *miscellaneous* instructions. The register type of instructions, which are shown in Figure 2(a), are one word instructions designed to perform an arithmetic, logic or shift operations, which are indicated by the opcode, on two operands, each of which are stored in the registers indicated by the `Src1` and `Src2` fields. The result of this operation will be returned to register indicated by the `Dest` field of the instruction.

The memory instructions, shown in Figure 2(b), are load and store instructions, which are designed to move data between a given register in the register file and memory. The memory address is specified by the second instruction word, where as the register address can be specified either by the `Dest` field, in the case of load instructions or by the `Src1` field, in the case of store instructions. The memory instructions can support four different addressing modes, including *immediate*, *direct*, *relative* and *indirect* addressing modes. In *relative* mode, the offset is stored in the register indicated by the `Src2` field of the instruction.

As shown in Figure 2(c), control instructions also comprise two words and can specify either jump, branch, subroutine call or subroutine return instructions. When the processor executes the jump instruction, for example, it loads the PC with jump address specified in the second word of the jump instruction and executes the instruction at the jump address in the next instruction cycle. The branch instruction has the same effect if the appropriate bit in the status register is 1; otherwise, the processor executes the next instruction in sequence. The six relation bits correspond to the six relational operations: equal, greater than, greater than or equal to, less than, or equal to, and not equal. These bits are set or reset by the miscellaneous instructions after comparing the contents of two registers.

Finally, miscellaneous instructions, which are shown in Figure 2(d), include the No-op instructions as well as those instructions necessary for setting and resetting particular registers in the datapath.The most important instruction in this group is the `Lstat` instruction, which is designed to compare the values in the registers indicated by the `Src1` and `Src2` fields and to set the six relational bits in the status register accordingly. As mentioned earlier, each branch instruction tests a specific bits after it has been set by the `Lstat`instructions.

3.1 Instruction Set Super FSMD

The instruction set completely specifies the behavior of a processor, in this sense, it can be thought of as a behavioral description of a processor. We now describe the instructions set in instruction set super FSMD, which describes the execution of all instructions. The super FSMD specifies nothing but the behavior of the processor and no architectural details are implied beyond the existence of a memory(*Mem*), a program counter(*PC*), an instruction register(*IR*), a register file(*RF*) and a status register(*Status*).

The instructions set super FSMD does not consider any timing constraints,data dependency or clock cycle duration. It gives the order in which the operations specified by each instruction will be executed.The source code for instruction set super FSMD is included in appendix A.

The instruction set super FSMD is shown in Figure 3. Each instruction has been specified in two parts. In the first part, which applies to all instructions, the processor fetches the instruction into the IR and increments the PC. In the second part, the processor decodes the type field to determine the instruction type and then executes the instruction by computing an effective address (EA), performing the operation specified by the opcode, and incrementing the PC in the case of memory and control instructions.

3.2 RTL-level library components

Our tool is used in the register transfer level synthesis. The datapath components are taken from a RTL library that maps these components to their gate level equivalence. The library also stores the delay parameters associated with each component. The delay parameter is the critical path (in ns) of the component.

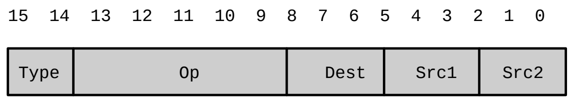
These RTL library components include:

- Storage units:register, register file,memory;
- Function units: ALU, Shifter;
- Interconnection: bus

The allocation of these resources is made from the component library. Table 3.2 is the library components used in our processor synthesis and the source code for these library components can be find at appendix A.1.

(a) Register instructions

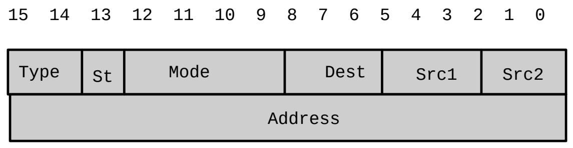
arithmetic, logic move and shift



Name	Action
Op Dest,Src1,Src2	RF(Dest)<-RF(Src1) Op RF(Src2)

(b) Memory instructions

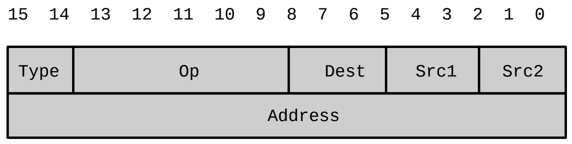
load and store



Name	Action
L imm Dest	RF[Dest]<-Adress
L dir Dest	RF[Dest]<-Mem[Address]
L rel Dest,Src2	RF[Dest]<-Mem[RF[Src2]+Address]
L in Dest	RF[Dest]<-Mem[Mem[Address]]
S dir Src1	Mem[Address]<-RF[Src1]
S rel Src1,Src2	Mem[RF{Src1]+Address]<-RF[Src1]
S in Src1	Mem[Mem[Address]]<-RF[Src1]

(c) Control instructions

jump, branch, call and return



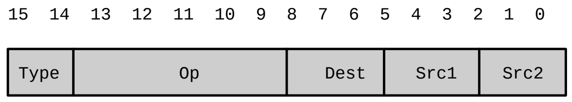
Name	Action
Jump Address	PC<-Address
Brel Address	[PC<-PC+1 if Status[rel]=0 PC<-Address if Status[rel]=1]

Call Address, Src1	Mem[Src1]<-PC+1; PC<-Addres;
	RF[Src1]<-RF[Src1]+1

Return	RF[Src1]<-RF[Src1]-1; PC<-Mem[Src1]

(d) Miscellaneous instructions

no-op, clear, status, set, and reset



Name	Action
No-op	Do nothing
Clear Dest	RF[Dest]<-0
Lstat Src1,Src2	Status<-R[Src1] $\overset{>}{\underset{<}{=}}$ Rf[Src2]
Sstat Dest	Status[Dest]<-1
Rstat Dest	Status[Dest]<-0

Figure 2. Instruction set of a 16-bit processor

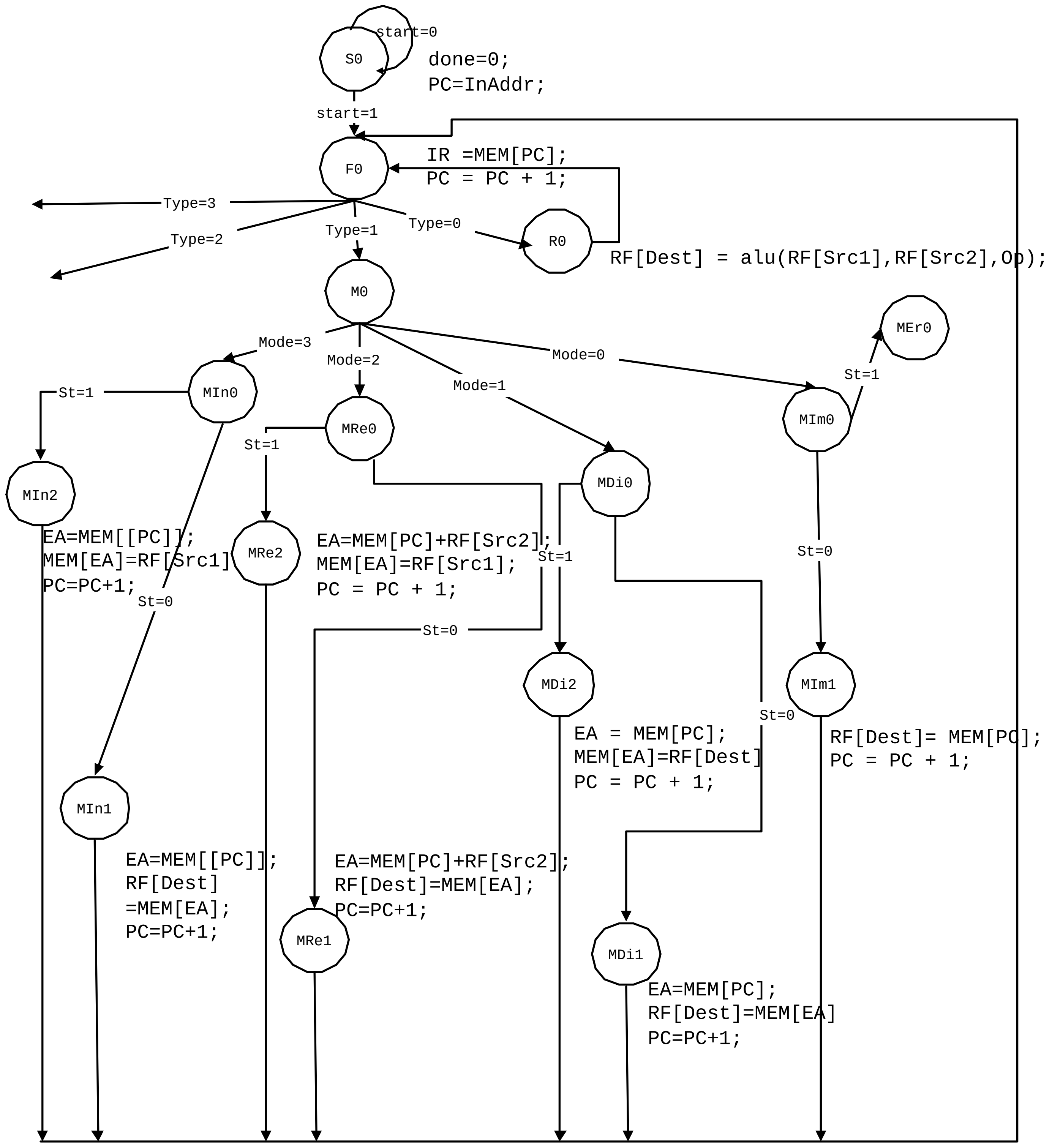


Figure 3. Instruction set super FSMD

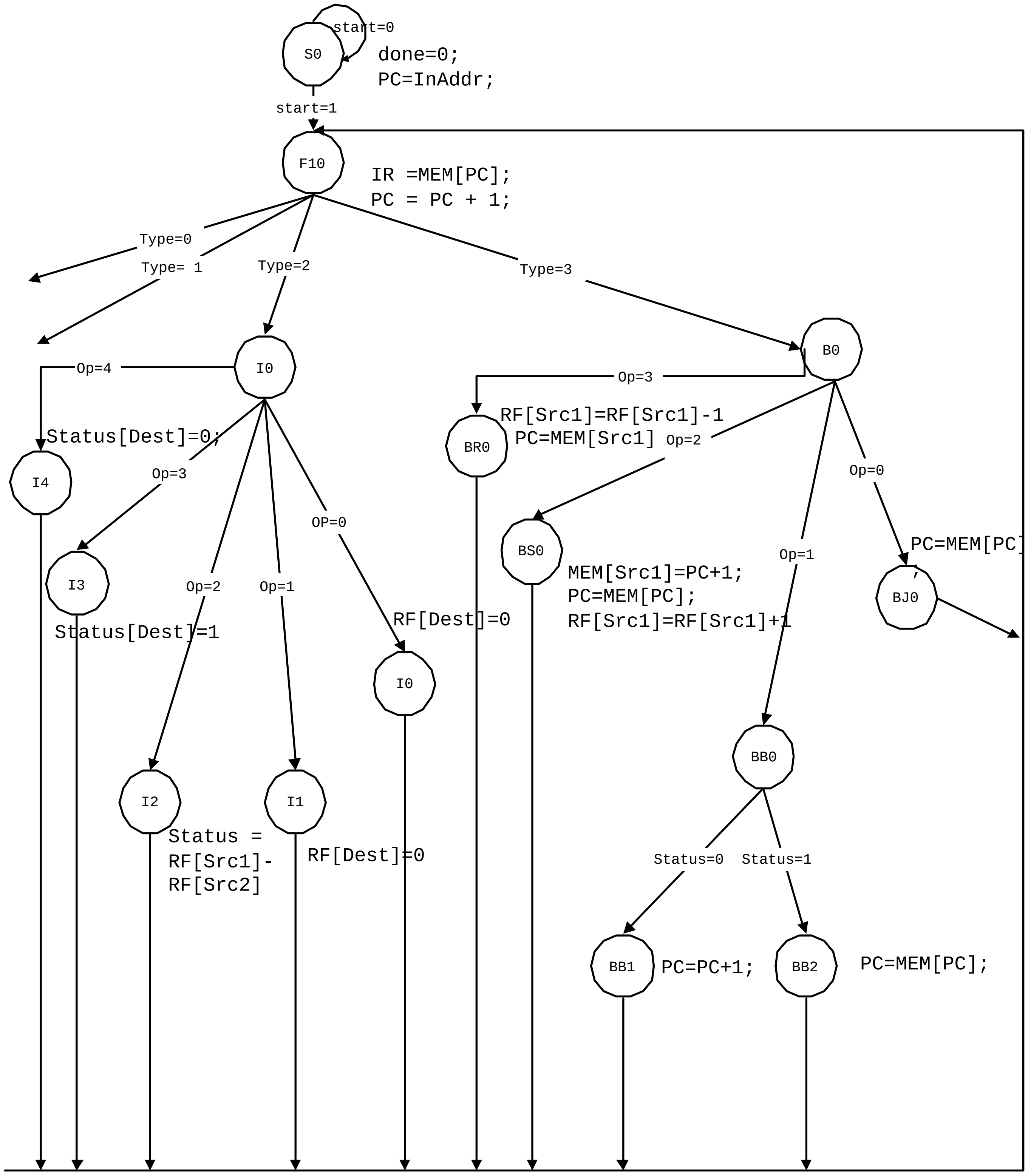
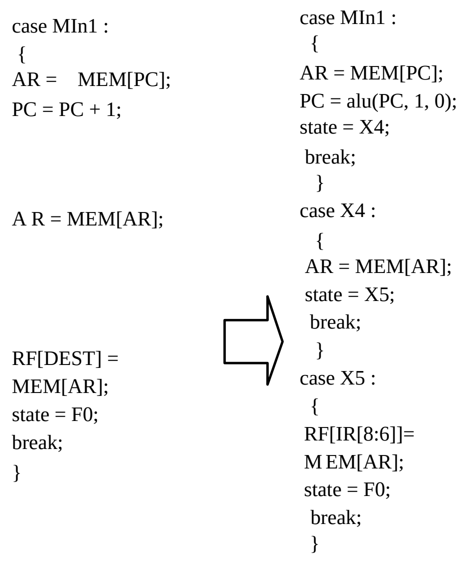


Figure 4. Instruction set super FSMD(cntl'd)

Resource Unit	Operations	Delays(ns)
ALU	add, sub, negate, and, or, not	3.02
ALU (pipelined)	add, sub, negate, and, or, not	3.02 1.5
Shifter	shl, shr	2.25
Register	register read	0.73
Register(setup)	register setup	0.59
RF	register file read	1.46
RF(setup)	register file setup	1.20
MEM	storage access	0.75
MUX	multiplexor	0.75
Latch	latch read	0.75
Latch(setup)	latch setup	0.59
CL	control logic	1.4

Table 1. RTL components delays



```
case MIn1 :
 {
AR =    MEM[PC];
PC = PC + 1;

A R = MEM[AR];

RF[DEST] =
MEM[AR];
state = F0;
break;
}
```

```
case MIn1 :
  {
AR = MEM[PC];
PC = alu(PC, 1, 0);
state = X4;
 break;
  }
case X4 :
  {
 AR = MEM[AR];
 state = X5;
  break;
  }
case X5 :
  {
RF[IR[8:6]]=
M EM[AR];
state = F0;
 break;
 }
```

Figure 5. State splitting by data dependency

4. Experimental Results

The input to the tool is a behavior description of the processor in RTL style 1(Appendix A.2). In the input source code, we explicitly define the super FSMD states in the declaration and use a `case` statement in a `while()` loop to move from state to state. In the design exploration process,we make allocation of different types or number of register files, ALUs, buses and try different kind of target RTL structure, the tool will generate different implementations.

We now discuss the performance of different implementations in detail.

4.1 Design 1: Datapath with Special Purpose Registers

In this implementation, the input resource combination to our tool include : one ALU, one shifter, one register file, five internal buses and several special registers for the target architecture: a program counter (*PC*), an instruction register (*IR*),a status register (*Status*), an address register (*AR*), and a data register (*DR*). The input resource also includes 64k of memory. We have eight registers in a register file, and the register file has two read ports and one write port.

Appendix B.2 shows the output result in RTL style 4 after synthesizing this design.As we can find in the output result, we have 12 extra states (denoted by X0-X11 in the output code).The reason why there are 12 extra states generated by our tool is that there is data dependency inside some states, so we must split these states. An example is shown in Figure 5, where the state MIn1 is split into 3 states; also, if the resource requirement can't be satisfied in a state, it also need to be split into multiple states. In the synthesized datapath(Figure 6), the address ports of the register file is directly connected with instruction register (RF): RAA is connected with SRC1(5:3) field of IR, RAB is connected with SRC2(2:0) field of IR, RWA is connected with DEST(8:6) filed of IR. The enable ports of register file (REA, REB, WE) are connected with the control output.

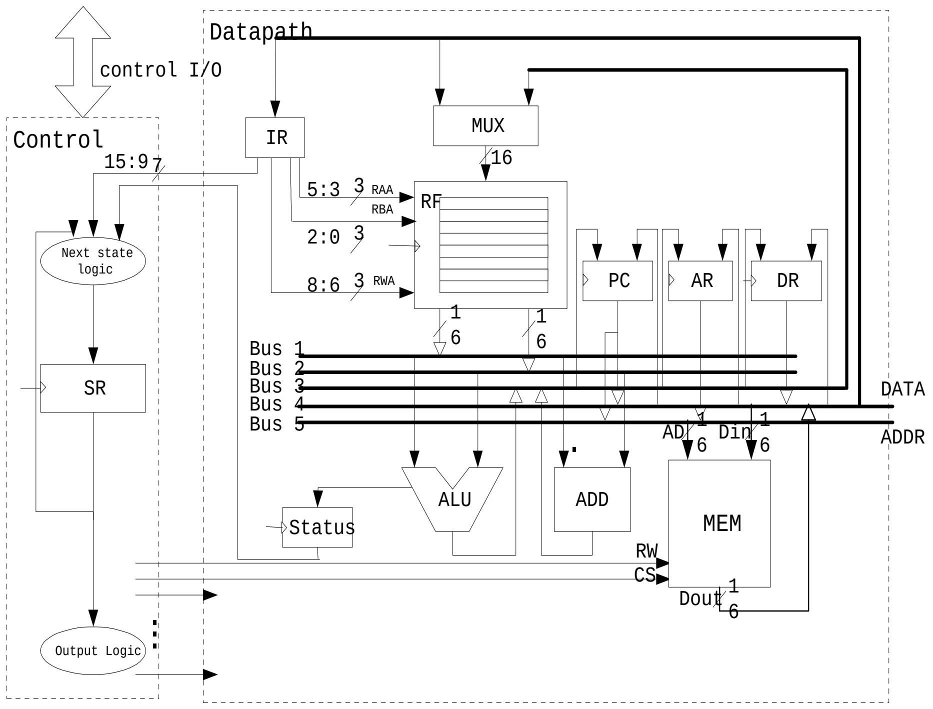
4.1.1 Performance Analysis

Performance metrics can be classified into three categories: clock cycles, control steps and execution times. Execution time is the final measure and the other two metrics contribute to its calculation. We define the execution time as the time interval needed to process a single instruction. If the number of clock cycles of for an instruction is `num_cycles` and the clock cycle delay is `clock_cycle`, the execution time can be computed as follows:

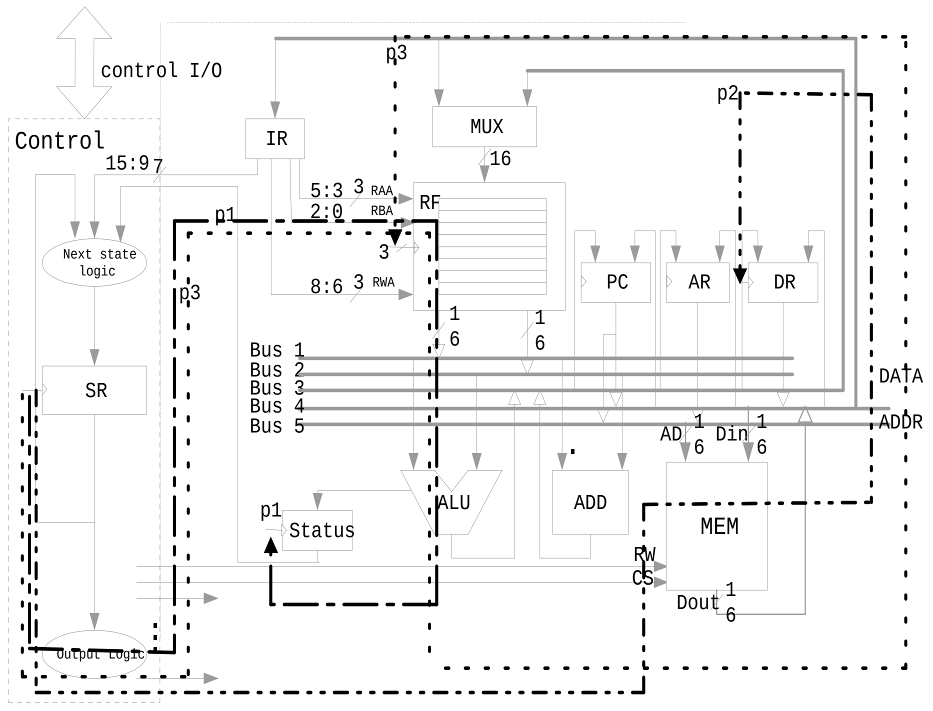
$$execution\_time = num\_cycles * clock\_cycle$$

The clock cycle of design one can be determined as the maximum of the critical path candidates as follows:

- Delay of path p1, computing the next state of the FSM: this path starts at state register(`SR`), goes through the control logic(`CL`), register file(`RF`), ALU and ends at the Status register(`Status`):
$$\begin{aligned}\Delta(p1) &= delay(SR) + delay(CL) + delay(RF) \\ &\quad + delay(ALU) + setup(Status) \\ &= 0.75 + 1.4 + 1.46 + 3.02 + 0.59 \\ &= 7.2ns\end{aligned}$$



(a) Datapath design with special purpose registers



(b) Critical path analysis

Figure 6. Design example one

- Delay of path p2, memory operations: for reading operations, the path starts at the state register(SR), goes through control logic, the memory and ends at the data register(DR):

$$\begin{aligned}\Delta(p2) &= delay(SR)+delay(CL)+delay(MEM)\\ &\quad setup(DR)\\ &= 0.75+1.4+2.6+0.59\\ &= 5.7ns\end{aligned}$$

- Delay of path p3, performing the arithmetic operation: this path starts at the state register(SR), goes through control logic, register file(RF), ALU, and ends at the register file(RF):

$$\begin{aligned}\Delta(p1) &= delay(SR)+delay(CL)+delay(RF)\\ &\quad +delay(ALU)+delay(MUX)+setup(RF)\\ &= 0.75+1.4+1.46+3.02+0.66+0.59\\ &= 7.9ns\end{aligned}$$

Here `delay(SR)` is the delay lapsed in reading state register `SR` which is the same as the `Register` in Table 3.2, `delay(CL)` is the delay of output logic in control unit, `delay(ALU)` is the delay of the ALU, `delay(RF)` is the delay of reading data from the register file `RF`, `setup(RF)` is the setup time of the register file `RF`, `setup(DR)` is the setup time of the data register(`DR`) connected to the memory read port, `delay (AR)` is the day of reading the address register `AR`, `delay(MEM)` is the delay of reading the memory, `delay(MUX)` is the delay of multiplexor before the input port of register file.

Hence, the minimum clock cycle is:

$$Clock\_cycle = max(\Delta(p1),\Delta(p2),\Delta(p3)) = 7.9ns$$

4.2 Design 2: Datapath with Register File only

In the second design, we will bind the special purpose registers (`PC`, `AR`,`DR`) into the register file: so we delete these special purpose registers from the input resource combination(library file), and our RTL tool binds these registers to entries in the register file automatically. The binding result is shown in Figure 7.

Appendix C shows the output result in style 4 RTL after synthesizing this design. Comparing to the input, we notice There's 18 extra states generated due to resource constraint.

The clock cycle can be determined as the maximum of the critical path candidates as follows:

- Delay of path p1, computing the next state of the FSM:

$$\begin{aligned}\Delta(p1) &= delay(SR)+delay(CL)+delay(RF)\\ &\quad +delay(ALU)+setup(Status)\\ &= 0.75+1.4+1.46+3.02+0.59\\ &= 7.2ns\end{aligned}$$

- Delay of path p2, performing the memory operations:

$$\begin{aligned}\Delta(p2) &= delay(SR)+delay(CL)+delay(MEM)\\ &\quad +setup(RF)\\ &= 0.75+1.4+2.6+0.59\\ &= 5.3ns\end{aligned}$$

- Delay of path p3, performing the arithmetic operation:

$$\begin{aligned}\Delta(p1) &= delay(SR)+delay(CL)+delay(RF)\\ &\quad +delay(ALU)+setup(RF)\\ &= 0.75+1.4+1.46+3.02+0.59\\ &= 7.2ns\end{aligned}$$

Hence, the minimum clock cycle is:

$$Clock\_cycle = max(\Delta(p1),\Delta(p2),\Delta(p3),\Delta(p4)) = 7.2ns$$

4.3 Design 3: Datapath with latched register file

We use pipeline in the datapath design in order to reduce the delay on the critical path. The first pipelined datapath design is shown in Figure 8. In this design, we add two latches to the output port of register file. By using latched register file, the longest path(p3) of design one is split into two paths in design three: p1 and p3. While the delay of path p2 remains same as that of design one. The delay of other paths are calculated as follows:

- Delay of path p1, which goes from the state register(SR) to the register file latch:

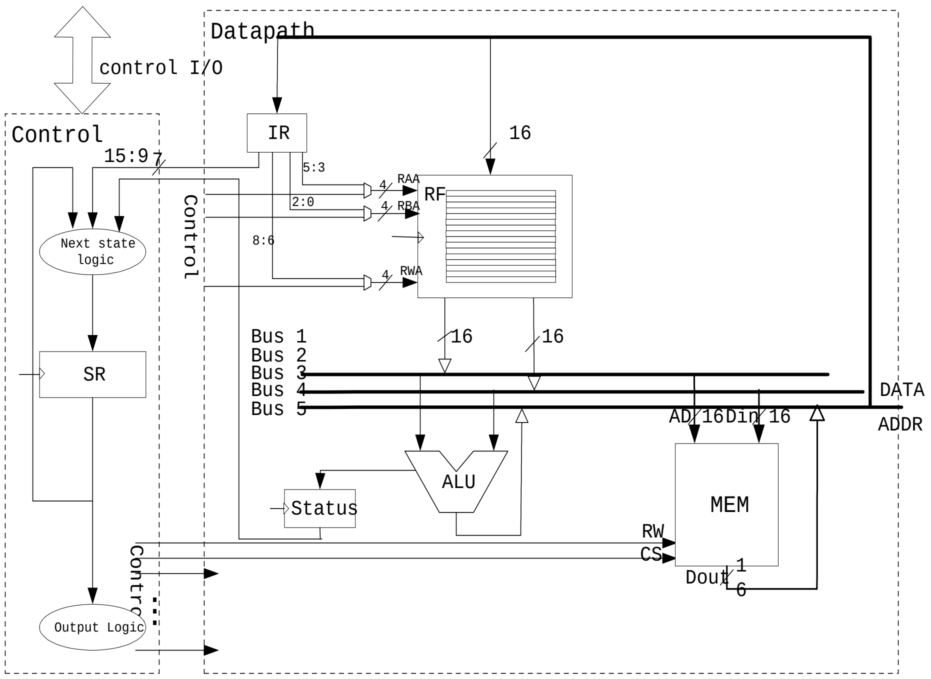
$$\begin{aligned}\Delta(p1) &= delay(SR)+delay(CL)+delay(RF)\\ &\quad +setup(Latch)\\ &= 0.75+1.4+1.46+0.59\\ &= 4.2ns\end{aligned}$$

- Delay of path p3, which starts at the register file latch, goes through ALU, MUX and finally ends at the register file(RF):

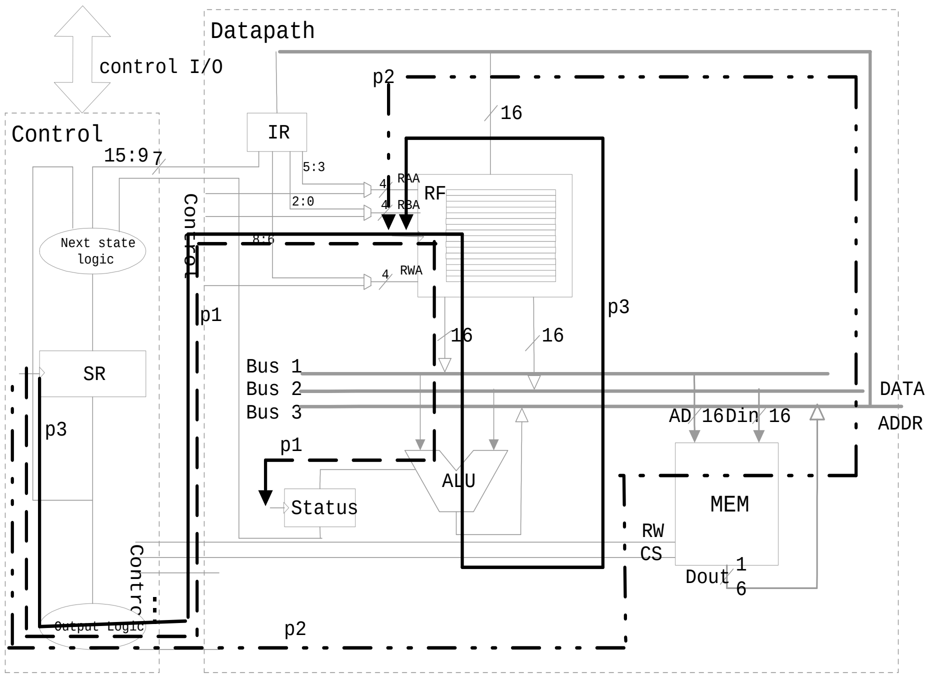
$$\begin{aligned}\Delta(p3) &= delay(Latch)+delay(ALU)+delay(MUX)\\ &\quad +setup(RF)\\ &= 0.75+3.02+0.66+0.59\\ &= 5ns\end{aligned}$$

- Delay of path p4, which starts at the register file latch, goes through ALU, and ends at the status register(`Status`):

$$\begin{aligned}\Delta(p4) &= delay(Latch)+delay(ALU)+setup(Status)\\ &= 0.75+3.02+0.59\\ &= 4.3ns\end{aligned}$$

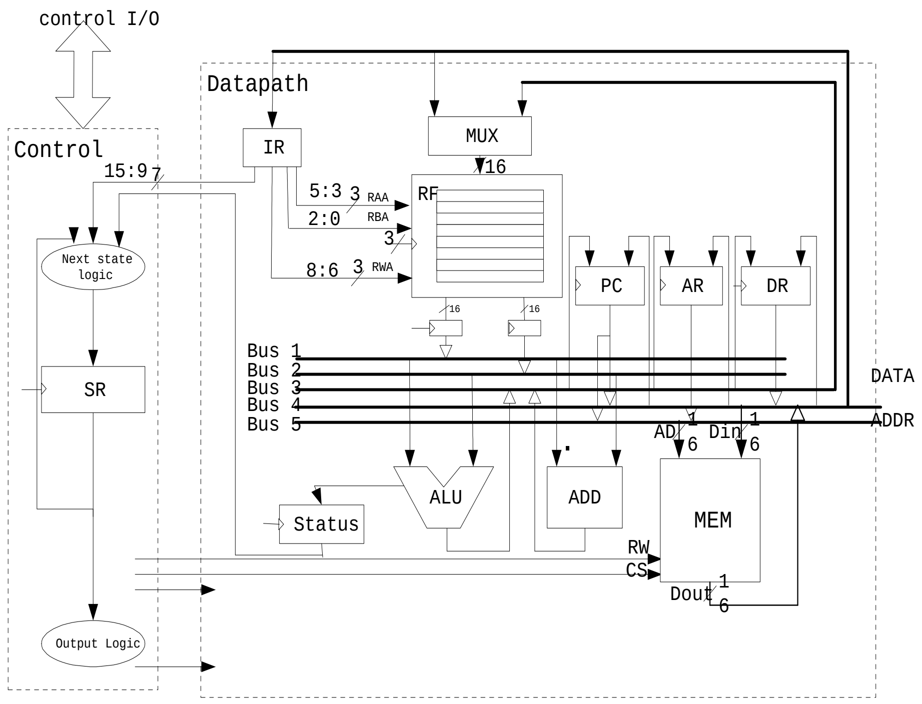


(a) Datapath design with register file only

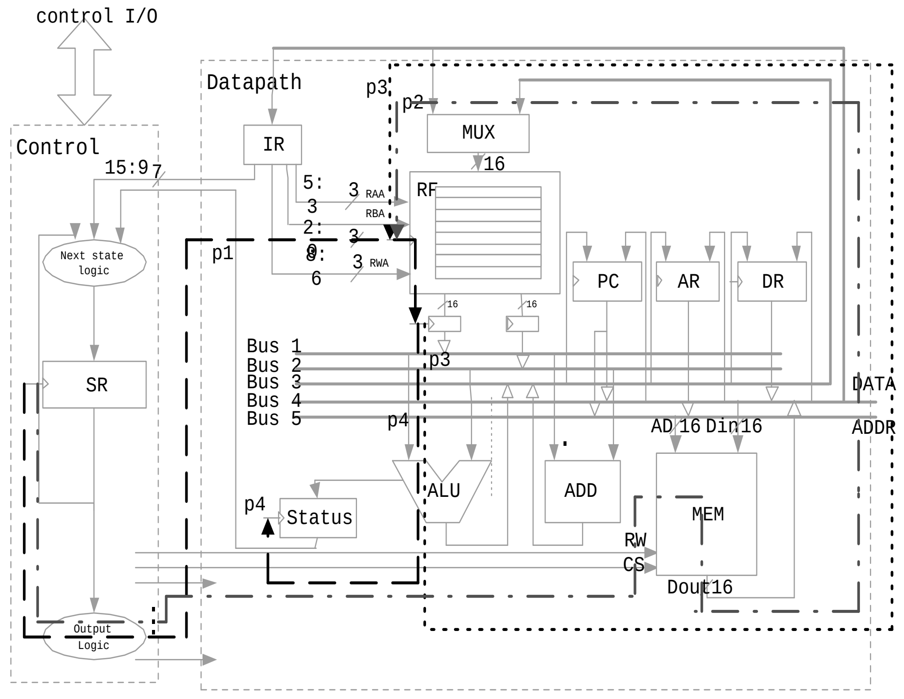


(b) Critical path analysis

Figure 7. Design example two

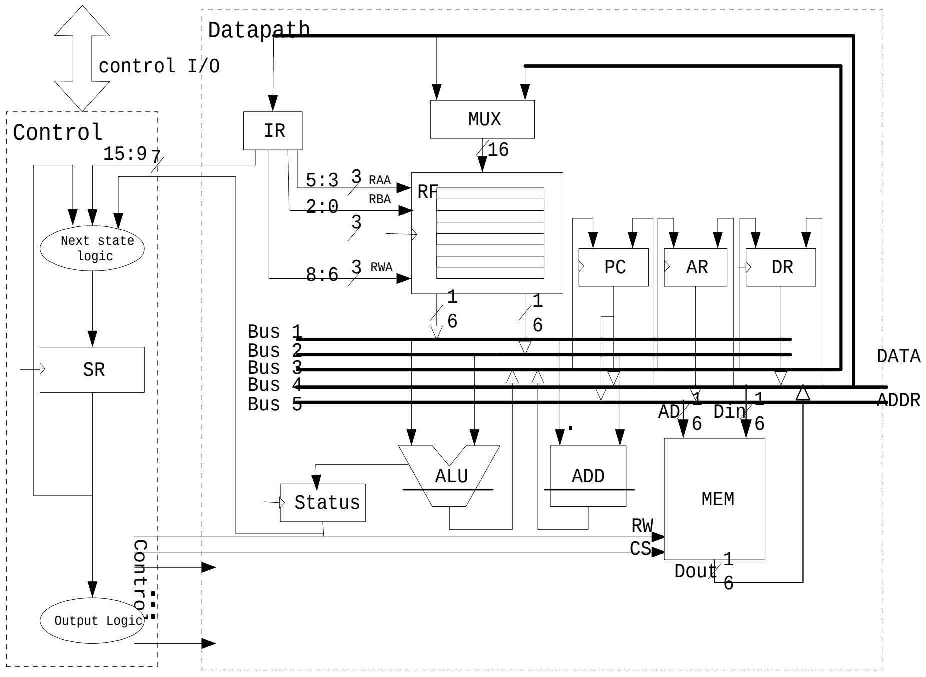


(a) Datapath design with latched register file

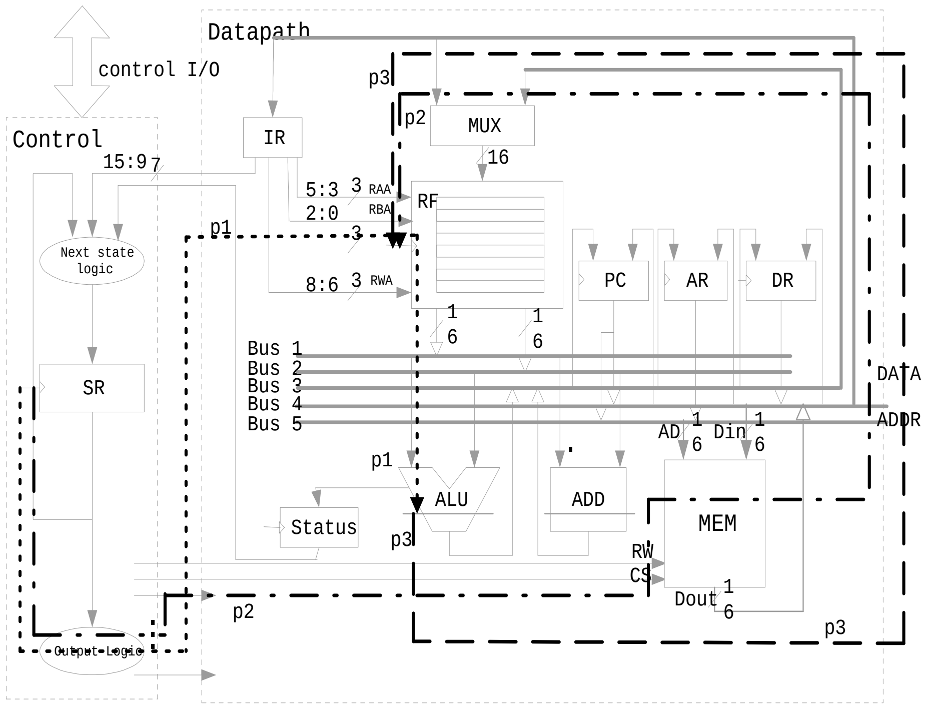


(b) Critical path analysis

Figure 8. Design example three



(a) Datapath design with pipelined functional unit



(b) Critical path analysis

Figure 9. Design example four

Hence, the minimum clock cycle is:

$$Clock\_cycle = max(\Delta(p1), \Delta(p2), \Delta(p3), \Delta(p4)) = 6ns$$

4.4 Design 4: Datapath with pipelined functional unit

In this design we attempt a pipelined implementation with a limited number of resources for further improvement in the performance. We allocate a pipelined ALU and a pipelined Shifter, other resources being the same as in Design 4.1

The result is shown in Figure 9. In this design, we pipeline the functional unit. By using pipelined functional unit, the longest path(p3) of design one is split into two paths in design four: p1 and p3.

Delay of path p2 remains same as that of design one. The delay of other paths are calculated as follows:

- Delay of path p1, which starts at the state register(t SR), goes through the control logic(CL), register file(RF), and ends at the register file latch:
$$\begin{aligned}\Delta(p1) &= delay(SR) + delay(CL) + delay(RF)\\ &+ setup(Latch)\\ &= 0.75 + 1.4 + 1.46 + 0.59\\ &= 4.2ns\end{aligned}$$
- Delay of path p3,which starts at the pipelined ALU, goes through MUX and ends at the register file(RF):
$$\begin{aligned}\Delta(p3) &= pipe(ALU) + delay(MUX) + setup(RF)\\ &= 1.5 + 0.66 + 1.2\\ &= 2.8ns\end{aligned}$$
- Delay of path p4,which goes from the pipelined ALU to the register file(RF):
$$\begin{aligned}\Delta(p4) &= pipe(ALU) + setup(Status)\\ &= 1.5 + 0.59\\ &= 2.1ns\end{aligned}$$

Pipe(ALU) is the delay of the pipelined ALU and is a half of a normal ALU delay as in Table 3.2. Since p1 has the largest delay among all the three candidates, the minimum clock cycle is:

$$Clock\_cycle = max(\Delta(p1), \Delta(p2), \Delta(p3), \Delta(p4)) = 6ns$$

4.5 Design 5: Datapath with multicycle memory

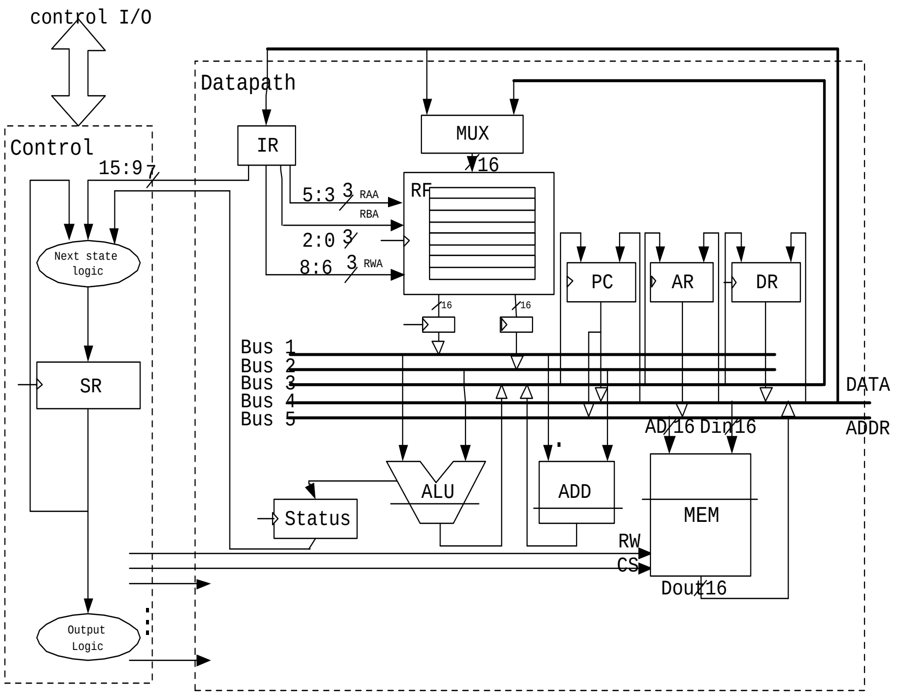
In the previous two designs, the maximum delay is on the path for memory operations. To reduce the delay on the critical path, we use multicycle memory in the datapath design, also we use both pipelined functional unit and latched register file in the design. The revised datapath is shown in figure 10.

The path delay is calculated as follows:

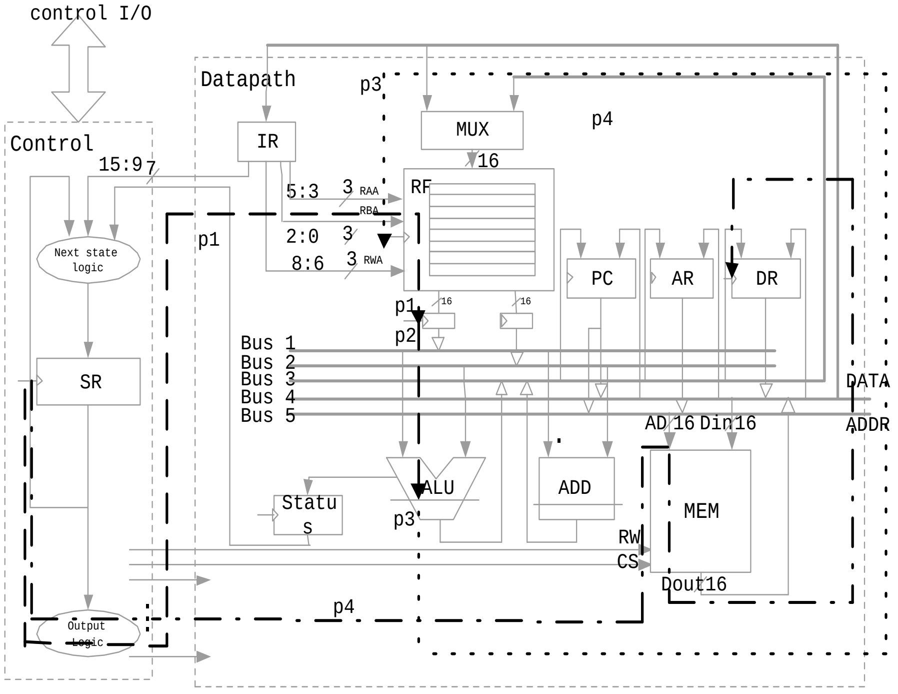
- Delay of path p1, which starts at the state register(SR), goes through the control logic(CL), register file(RF), and ends at the register file latch:
$$\begin{aligned}\Delta(p1) &= delay(SR) + delay(CL) + delay(RF)\\ &+ setup(Latch)\\ &= 0.75 + 1.4 + 1.46 + 0.59\\ &= 4.2ns\end{aligned}$$
- Delay of path p2, which goes from the register file to pipelined ALU:
$$\begin{aligned}\Delta(p2) &= delay(Latch) + pipe(ALU)\\ &= 0.75 + 1.5\\ &= 2.3ns\end{aligned}$$
- Delay of path p3, starts from pipelined ALU, goes through MUX, and ends at the register file(RF) to finish the computation of the arithmetic operations:
$$\begin{aligned}\Delta(p3) &= pipe(ALU) + delay(MUX) + setup(RF)\\ &= 1.5 + 0.66 + 0.59\\ &= 2.8ns\end{aligned}$$
- Delay of path p4, memory operations: for reading operations, the path starts at the state register(SR), goes through control logic, the memory and ends at the data register(DR):
$$\begin{aligned}\Delta(p4) &= delay(SR) + delay(CL) + delay(MEM)\\ &+ delay(MUX) + setup(RF)\\ &= 0.75 + 1.4 + 2.6 + 0.66 + 0.59\\ &= 6.0ns\end{aligned}$$
Since it is a multicycle memory, the memory operation takes two clock cycles. So the clock cycle time of each memory operation would be: 1/2 * $\Delta(p4) = 3.0ns$

Hence, the minimum clock cycle is:

$$Clock\_cycle = max(\Delta(p1), \Delta(p2), \Delta(p3), 1/2 * \Delta(p4) = 4.2ns$$



(a) Datapath design with multi-cycle memory



(b) Critical path analysis

Figure 10. Design example five

	Instruction execution time of different designs (ns)				
Design	One	Two	Three	Four	Five
Register	15.8	14.4	18	18	16.8
Memory	47.4	50.4	36	42	33.6
Control	39.5	43.2	30	30	25.2
Misc	23.7	21.6	18	18	13.2
Clk Period	7.9	7.2	6	6	4.2

Table 2. Instruction execution time of different designs

4.6 Instruction execution time of different designs

Table 4.6 is the instruction execution time of using different datapath for the processor. As we can see from this table, design two takes the shortest execution time for the register instructions; for other kind of instructions, design five is the best, it takes the shortest instruction execution time. Design two is worst, it takes longest execution time for all the instructions.

Also, we notice that there are two ways to improve the design performance, increasing the number of resources used in the design or introducing pipelined units in the design. Employing more resources in the design can reduce the number of states in the FSMD of the behavior with little change in the critical path. Introduction of pipelined units in the design causes a drastic reduction in clock cycle but at the same time there's more states generated and the total number of execution cycles increases, some times this leads to poorer performance.

5. Conclusion and Future Works

In this report, we presented the super FSMD of a simple 16 bit microprocessor and used our RTL synthesis tool to generate the different kind of datapath for this microprocessor. The first design is a non-pipelining implementation: we use special purpose registers and single stage ALU, register file, memory, etc to build a datapath based the given FSMD specification. It is a cheap and straightforward implementation. Compared with the pipelined version, the performance of this design is poor, but the cost is low and the architecture is easy to implement.

In the second design, we try to bind the special purpose registers into register file. By doing this, we can replace the expensive registers with the lower cost register file, as a result this design leads to poor performance: while its clock period remains nearly same as the first design, there's more clocks needed to execute the individual instructions.

In the first design, the critical path is for performing the arithmetic operation, we can reduce the path length by inserting latches or using pipelined functional unit. The last three designs are improved implementations of the first design: in the third design, we use datapath pipeline, where we use latched register file; in the fourth design, we replace the ALU and SHIFT with the pipelined implementation version. These two designs have a shorter critical path, therefore, their clock period is shorter than the first design.

In the fifth design, the memory is changed to multicycle memory, so the critical path length can be further reduced. We have the shortest clock period among all the five designs.

We demonstrate the different implementations of the 16 bit microprocessor, they are generated by our RTL synthesis tool with different allocation of resources from the component library. Based on these design, we make comparative analysis of their performance. The result allow the end user to decide upon the final implementation which strikes out an optimal balance between the cost and the performance.

However, there are still some impending modifications in the tool. Our approach introduces storage units like register file and memory in the component library mapping of whose ports is not supported in the current binding algorithm. The future extension to our work is proposing a binding algorithm which considers the mapping of ports of the storage units. We expect to release the improvised algorithm in the future. Also ,we are working on the exact syntax for pipelined/multicycle operations for our RTL input/output code.After we make a decision of the syntax, we will append the output code in the appendix.

References

[Acc01] Accellera C/C++ Working Group. RTL Semantics:Draft Specification. Feburary 2001.

[Gaj97] D. Gajski. *Principles of Digital Design*. Pretence Hall, 1997.

[GDLW92] D. Gajski, N. Dutt, S. Lin, and A. Wu. *High Level Synthesis: Introduction to Chip and System Design*. Kluwer Academic Publishers, 1992.

[ZSY$^+$00] P. Zhang, D. Shin, H. Yu, Q. Xie, and D. Gajski. SpecC RTL Design Methodology. Technical Report ICS-TR-00-44, University of California, Irvine, December 2000.

A. Instruction Set Simulator in RTL style 1

A.1 RTL component Library

```
bit[31:0] alu(bit[31:0] a, bit[31:0] b, bit[2:0] ctrl)
{
    note  alu.library = "1";
    note  alu.a="data";
    note  alu.b="data";
    note  alu.sum="data";
    note  alu.ctrl="control";

    note  alu.type="rca";
    note  alu.width="472";
    note  alu.height="920";
    note  alu.cost="100";
    note  alu.pipelined = "0";
    note  alu.delay="1";
    note  alu.bits="32";
    note  alu.operation="+,-,<,<=,>,>=,!=,==,&,+:,-:,+=,-=";
    note  alu.num_wports= "2";
    note  alu.num_rports = "1";
    bit[31:0] sum;

    switch(ctrl)   {
        case 000b:  // +
            sum = a+b;
            break;
        case 001b:  // -
            sum = a-b;
            break;
        case 010b:  // <
            sum = (a<b)? 0x0001:0x0000;
            break;
        case 011b:  // <=
            sum = (a<=b)? 0x0001:0x0000;
            break;
        case 100b:  // >
            sum = (a>b)? 0x0001:0x0000;
            break;
        case 101b:  // >=
            sum = (a>=b)? 0x0001:0x0000;
            break;
        case 110b:  // !=
            sum = (a!=b)? 0x0001:0x0000;
            break;
        case 111b:  // ==
            sum = (a==b)? 0x0001:0x0000;
            break;
        case 1000b:     // &
            sum = a&b;
            break;
    }
    return sum;
}

bit[31:0] add(bit[31:0] a, bit[31:0] b, bit[2:0] ctrl)
{
    note add.library = "1";
    note add.si = "data";
    note add.amount = "data";
    note add.so = "data";
    note add.ctrl = "control";

    note add.type = "adda";
    note add.width = "272";
    note add.height = "420";
    note add.cost = "60";
    note add.pipelined = "0";
    note add.delay = "1";
    note add.bits = "32";
    note add.operation = "+";
    note add.num_wports= "2";
    note add.num_rports = "1";

    bit[31:0] so;
    so = a+b;
    return so;
}

void RF(event clk, bit[0:0] rst, bit[31:0] inp,
    bit[1:0] raA, bit[1:0] raB, bit[0:0] reA, bit[0:0] reB,
    bit[1:0] wa, bit[0:0] we, bit[31:0] outA, bit[31:0] outB)
{
    note RF.library = "1";
    note RF.type = "RF";
    note RF.size = "8";
    note RF.width = "272";
    note RF.height = "420";
    note RF.cost = "60";
    note RF.pipelined = "0";
    note RF.delay = "0";
    note RF.num_inports= "1";
    note RF.num_outports = "2";
    note RF.bits = "32";
}

void PC(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note PC.library = "1";
    note PC.type = "reg";
    note PC.size = "1";
    note PC.width = "100";
    note PC.height = "220";
    note PC.cost = "30";
    note PC.pipelined = "0";
    note PC.delay = "0";
    note PC.num_inports= "1";
    note PC.num_outports = "1";
    note PC.bits = "32";
}


void Status(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note Status.library = "1";
    note Status.type = "reg";
    note Status.size = "1";
    note Status.width = "100";
    note Status.height = "220";
    note Status.cost = "30";
    note Status.pipelined = "0";
    note Status.delay = "0";
    note Status.num_inports= "1";
    note Status.num_outports = "1";
    note Status.bits = "32";
}


void IR(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note IR.library = "1";
    note IR.type = "reg";
    note IR.size = "1";
    note IR.width = "100";
    note IR.height = "220";
    note IR.cost = "30";
    note IR.pipelined = "0";
    note IR.delay = "0";
    note IR.num_inports= "1";
    note IR.num_outports = "1";
    note IR.bits = "32";
}

void AR(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note AR.library = "1";
    note AR.type = "reg";
    note AR.size = "1";
    note AR.width = "100";
    note AR.height = "220";
    note AR.cost = "30";
    note AR.pipelined = "0";
    note AR.delay = "0";
    note AR.num_inports= "1";
    note AR.num_outports = "1";
    note AR.bits = "32";
}
```

```
void DR(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note DR.library = "1";
    note DR.type = "reg";
    note DR.size = "1";
    note DR.width = "100";
    note DR.height = "220";
    note DR.cost = "30";
    note DR.pipelined = "0";
    note DR.delay = "0";
    note DR.num_inports= "1";
    note DR.num_outports = "1";
    note DR.bits = "32";
}

void MEM(event clk, bit[0:0] rst, bit[31:0] inp,
    bit[1:0] raA, bit[1:0] raB, bit[0:0] reA, bit[0:0] reB,
    bit[1:0] wa, bit[0:0] we, bit[31:0] outA, bit[31:0] outB)
{
    note MEM.library = "1";
    note MEM.type = "mem";
    note MEM.size = "65536";
    note MEM.width = "272";
    note MEM.height = "420";
    note MEM.cost = "60";
    note MEM.pipelined = "0";
    note MEM.delay = "1";
    note MEM.num_rwports= "1";
    note MEM.num_aports = "1";
    note MEM.bits = "32";
}

void bus(bit[31:0] outp, bit[31:0] inp)
{
    note bus.library = "1";
    note bus.type = "bus";
    note bus.width = "1";
    note bus.height = "1";
    note bus.cost = "60";
    note bus.delay = "0";
    note bus.bits = "32";
}
```

A.2 Instruction Set Simulator

```
/*********************************************************
* SpecC code for an Instruction Set Simulator
* Author: Haobo Yu
*         Center for Embedded Computer Systems
*         University of California, Irvine
* Time  : Sep,25,2001
*********************************************************/
#include<stdio.h>
import "lib";


#define TYPE IR[15:14]
#define OP   IR[13:9]
#define MODE IR[12:9]
#define ST IR[13:13]
#define DEST IR[8:6]
#define SRC1 IR[5:3]
#define SRC2 IR[2:0]

behavior ISS(in event clk, in bit[0:0] rst, in bit[15:0] InAddr,
out bit[15:0] OutAddr,in bit[0:0] start, out bit[0:0] done)
{

    note ISS.scheduled = "0";
    note ISS.fubind = "0";
    note ISS.regbind = "0";
    note ISS.busbind = "0";

    note ISS.clk = "clk";
    note ISS.rst = "rst";
    note ISS.InAddr = "data";
    note ISS.start = "ctrl";
    note ISS.done = "ctrl";

    void main(void)
    {
        bit[15:0] PC;
        bit[15:0] IR;
        bit[15:0] Status;
        bit[15:0] RF[8];
        bit[15:0] AR;
        bit[15:0] DR;
        bit[15:0] MEM[65536];


        enum state { S0,F0,R0,M0,MIm0,MIm1,MDi0,MDi1,MDi2,MRe0,MRe1,
        MRe2,MIn0,MIn1,MIn2,MEr0,B0, BJ0,BB0,BB1,BB2,BS0,BS1,BR0,BR1,
        BEr0,I0,I1,I2,I3,I4,IEr0 } state;
        while (1)
        {
            wait(clk);
            if (rst)
            {
                state = S0;
            }
            switch (state)
            {
                //reset state
                case S0 :
                {
                    done = 0;
                    PC = InAddr;
                    if (start!=0)
                    {
                        state = F0;
                    }
                    else
                    {
                        state = S0;
                    }
                    break;
                }

                //Instruction Fetch
                case F0 :
                {
                    IR = MEM[PC];
                    PC = add(PC,1,0);
                    switch(TYPE) {
                    case 0:
                     state = R0;
                     break;
                    case 1:
                     state = M0;
                     break;
                    case 2:
                     state = B0;
                     break;
                    case 3:
                     state = I0;
                     break;
                    }
                    break;
                }

                //Register Instructions
                case R0 :
                {
                    RF[DEST] = alu(RF[SRC1],RF[SRC2],OP);
                    state = F0;
                    break;
                }

        //Memory Instructions
```

```
        case M0 :
        {
            switch( MODE) {
                    case 0:
                     state = MIm0;
                     break;
                    case 1:
                     state = MDi0;
                     break;
                    case 2:
                     state = MRe0;
                     break;
                    case 3:
                     state = MIn0;
                     break;
                    }
                    break;
                }

                //Memory Instructions : Immediate
                case MIm0 :
                {
                    if ( ST == 0 )
                        state = MIm1;
                    else
                        state = MEr0;
                    break;
                }

                //Memory Instructions : Immediate
                case MIm1 :
                {
                    RF[DEST] = MEM[PC];
                    PC = add(PC,1,0);
                    state = F0;
                    break;
                }

                //Memory Instructions : Direct
                case MDi0 :
                {
                    if ( ST == 0 )
                        state = MDi1;
                    else
                        state = MDi2;
                    break;
                }

                //Memory Instructions : Direct  Load 1
                case MDi1 :
                {
                    AR = MEM[PC];
                    PC = add(PC,1,0);
                    RF[DEST] = MEM[AR];
                    state = F0;
                    break;
                }

                //Memory Instructions : Direct  Store 1
                case MDi2 :
                {
                    DR = RF[SRC1];
                    AR = MEM[PC];
                    PC = add(PC,1,0);
                    MEM[AR] = DR ;
                    state = F0;
                    break;
                }

                //Memory Instructions : Relative
                case MRe0 :
                {
                    if ( ST == 0 )
                        state = MRe1;
                    else
                        state = MRe2;
                    break;
                }

                //Memory Instructions : Relative  Load 1
                case MRe1 :
                {
                    RF[DEST] = MEM[PC];
                    PC = add(PC,1,0);
                    AR = RF[DEST] + RF[SRC2];
                    RF[DEST] = MEM[AR];
                    state = F0;
                    break;
                }

                //Memory Instructions : Relative  Store 1
                case MRe2 :
                {
                    RF[DEST] = MEM[PC];
                    PC = add(PC,1,0);
                    AR = RF[DEST] + RF[SRC2];
                    DR = RF[SRC1];
                    MEM[AR] = DR;
                    state = F0;
                    break;
                }

                //Memory Instructions : Indirect
                case MIn0 :
                {
                    if ( ST == 0 )
```

```
                state = MIn1;
            else
                state = MIn2;
            break;
        }

        //Memory Instructions : Indirect  Load 1
        case MIn1 :
        {
            AR = MEM[PC];
            PC = add(PC,1,0);
            AR = MEM[AR];
            RF[DEST] = MEM[AR];
            state = F0;
            break;
        }

        //Memory Instructions : Indirect  Store 1
        case MIn2 :
        {
            AR = MEM[PC];
            PC = add(PC,1,0);
            DR = RF[SRC1];
            AR = MEM[AR];
            MEM[AR] = DR;
            state = F0;
            break;
        }

        //Memory Instructions : Error State
        case MEr0:
        {
            state = S0;
            break;
        }

        //Branch Instrunctions
        case B0:
        {
    switch( OP) {
            case 0:
             state = BJ0;
             break;
            case 1:
             state = BB0;
             break;
            case 2:
             state = BS0;
             break;
            case 3:
             state = BR0;
             break;
            default:
```

```
                        state = BEr0;
                        break;
                    }
                    break;
                }

                //Branch Instrunctions : Jump
                case BJ0:
                {
                    PC = MEM[PC];
                    state = F0;
                    break;
                }

                //Branch Instrunctions : Branch
                case BB0:
                {
            if ( Status == 0 )
                state = BB1;
            else
                state = BB2;
            break;
        }

                //Branch Instrunctions : Branch 1
                case BB1:
                {
                    PC = add(PC,1,0);
                    state = F0;
                    break;
                }

                //Branch Instrunctions : Branch 2
                case BB2:
                {
                    PC = MEM[PC];
                    state = F0;
                    break;
                }

                //Branch Instrunctions : Subroutine 1
                case BS0:
                {
                    DR = MEM[PC];
                    AR = RF[SRC1];
                    PC = add(PC,1,0);
                    MEM[AR] = PC;
                    PC = DR;
                    RF[SRC1] = RF[SRC1] + 1;
                    state = F0;
                    break;
                }
```

```
//Branch Instrunctions : Return  1
case BR0:
{
    AR = MEM[AR];
    state = BR1;
    PC = MEM[AR];
    RF[SRC1] = RF[SRC1] + 1;
    state = F0;
    break;
}

//Branch Instructions : Error State
case BEr0:
{
    state = S0;
    break;
}

//Implied Instructions
case I0:
{
    switch( OP) {
    case 0:
     state = F0;
     break;
    case 1:
     state = I1;
     break;
    case 2:
     state = I2;
     break;
    case 3:
     state = I3;
     break;
    case 4:
     state = I4;
     break;
    default:
         state = IEr0;
         break;
    }
    break;
}

//Implied Instructions 1
case I1:
{
   RF[DEST] = 0;
   state = F0;
   break;
}

//Implied Instructions 2
```

```
                    case I2:
                    {
                       Status = RF[SRC1] - RF[SRC2];
                       state = F0;
                       break;
                    }

                    //Implied Instructions 3
                    case I3:
                    {
                       Status[DEST] = 1;
                       state = F0;
                       break;
                    }

                    //Implied Instructions 4
                    case I4:
                    {
                       Status[DEST] = 0;
                       state = F0;
                       break;
                    }

                    //Implied Instructions : Error State
                    case IEr0:
                    {
                state = S0;
                        break;
                    }
                }
            }
        }
};
```

A.3 Test Bench

```
/*****************************************************************************
*  Title: tb.sc
* Author: Haobo Yu
*         Center for Embedded Computer Systems
*         University of California, Irvine
*  Date: 02/03/2002
*  Description: testbench for instruction set simulator
*****************************************************************************/

import "clkgen";
import "iss";

behavior Main
{
    unsigned bit[15:0] InAddr, OutAddr;
    unsigned bit[0:0] rst;
    event clk;
    unsigned bit[0:0] start,done;

    CLKGEN U00(clk);
    IO     U01(clk, rst, InAddr,  start);
    ISS   U02(clk, rst, InAddr, start,done);
    int main (void)
    {
        par {
            U00.main();
            U01.main();
            U02.main();
        }
        return 0;
    }
};
```

A.4 Input/Output

```
/****************************************************************************
*  Title: io.sc
* Author: Haobo Yu
*         Center for Embedded Computer Systems
*         University of California, Irvine
*  Date: 02/03/2002
*  Description: input/output for testbench
****************************************************************************/

#include <stdio.h>
#include <stdlib.h>

unsigned bit[15:0] PC;
unsigned bit[15:0] IR;
unsigned bit[15:0] Status;
unsigned bit[15:0] RF[8];
unsigned bit[15:0] AR;
unsigned bit[15:0] DR;
unsigned bit[15:0] MEM[65536];

behavior IO(in event clk, out unsigned bit[0:0] rst,
    out unsigned bit[15:0] Address,out unsigned bit[0:0] Start)
{
    void main(void) {
        char buf[16];
        int  i;

        rst = 1b;
        Start = 0b;
        wait(clk);
        wait(clk);

        rst = 0b;       // deassign reset

        //initiliazation
        PC=0;
        IR=0;
        Status=0;
        AR=0;
        DR=0;
        for ( i=0;i<100;i++)
            MEM[i]=0;
        //Put Instructions to memory
        //The following code is to calculate the value of 15+9,15 is
        //in MEM[2],9 is in MEM[5],result should be in MEM[3]
        //Jump 4
        //Lim  #9,r0
        //Ldi @2,r1
        //add r2,r0,r1
        //Sin @11,@3
        //Jump 0
        MEM[0]=1000000000000000b;//Jump 4
        MEM[1]=0000000000000100b;//     ;Address
        MEM[2]=0000000000001111b;//     ;Data: 15
        MEM[3]=0000000000000000b;//     ;The result goes here
        MEM[4]=0100000000000000b;//Lim #9,r0    ;Load Immediate data to RF[0];
        MEM[5]=0000000000001001b;//     ;Data: 9
        MEM[6]=0100001001000000b;//Ldi @2,r1    ;Load Direct MEM[2] to RF[1]
        MEM[7]=0000000000000010b;//     ;Address 2
        MEM[8]=0000000010000001b;//add r2,r0,r1 ;ALU operation: RF[2]=RF[0]+RF[1];
        MEM[9]=0110011000010000b;//Sin @11,@3   ;Store Indirect to MEM[3]
            MEM[10]=0000000000001101b;//            ;Indirect Address: 13
            MEM[11]=1000000000000000b;//Jump 0
            MEM[12]=0000000000000000b;//        ;Address
            MEM[13]=0000000000000011b;//        ;Result Address:3



        printf("Register␣Dump:\n");
        printf("PC=0x%x␣IR=0x%x␣Status=0x%x␣␣␣AR=0x%x␣␣␣DR=0x%x␣\n",
        (int)PC,(int)IR,(int)Status,(int)AR,(int)DR);
        printf("RF[0]=%d,RF[1]=%d,RF[2]=%d,RF[3]=%d\n",
        (int)RF[0],(int)RF[1],(int)RF[2],(int)RF[3]);
        printf("RF[4]=%d,RF[5]=%d,RF[6]=%d,RF[7]=%d\n",
        (int)RF[4],(int)RF[5],(int)RF[6],(int)RF[7]);
        for(i=0;i<12;i++)
            printf("MEM[%d]:␣0x%x\n", i, (int)MEM[i]);
        Start = 1b;
        Address =0 ;
        while (1) {
            wait(clk);
            printf("Press␣any␣key␣to␣go␣:\n");
            gets(buf);
            printf("Register␣Dump:\n");
            printf("PC=0x%x␣IR=0x%x␣Status=0x%x␣␣␣AR=0x%x␣␣␣DR=0x%x␣\n",
            (int)PC,(int)IR,(int)Status,(int)AR,(int)DR);
            printf("RF[0]=%d,RF[1]=%d,RF[2]=%d,RF[3]=%d\n",
            (int)RF[0],(int)RF[1],(int)RF[2],(int)RF[3]);
            printf("RF[4]=%d,RF[5]=%d,RF[6]=%d,RF[7]=%d\n",
            (int)RF[4],(int)RF[5],(int)RF[6],(int)RF[7]);
            for(i=0;i<12;i++)
                if (i==3)
                    printf("MEM[%d]:␣0x%x␣<==result\n", i, (int)MEM[i]);
                else
                    printf("MEM[%d]:␣0x%x\n", i, (int)MEM[i]);
        }
    }
};
```

A.5 Clock Generator

```
/***************************************************************************
*  Title: clkgen.sc
* Author: Haobo Yu
*         Center for Embedded Computer Systems
*         University of California, Irvine
*  Date:  02/03/2002
*  Description: clock signal generator
***************************************************************************/
bit[15:0] alu(bit[15:0] a, bit[15:0] b, int ctrl)
#define clk_period 5
behavior CLKGEN(out event clk)
{
    void main(void) {
        while (1) {
            waitfor(clk_period);
            notify(clk);
        }
    }
};
```

B. Design 1: Datapath with special registers

B.1 Design 1 input: RTL component library

```
bit[31:0] alu(bit[31:0] a, bit[31:0] b, bit[2:0] ctrl)
{
    note  alu.library = "1";
    note  alu.a="data";
    note  alu.b="data";
    note  alu.sum="data";
    note  alu.ctrl="control";

    note  alu.type="rca";
    note  alu.width="472";
    note  alu.height="920";
    note  alu.cost="100";
    note  alu.pipelined = "0";
    note  alu.delay="1";
    note  alu.bits="32";
    note  alu.operation="+,-,<,<=,>,>=,!=,==,&,+:,-:,+=,-=";
    note  alu.num_wports= "2";
    note  alu.num_rports = "1";
    bit[31:0] sum;

    switch(ctrl)   {
        case 000b:  // +
            sum = a+b;
            break;
        case 001b:  // -
            sum = a-b;
            break;
        case 010b:  // <
            sum = (a<b)? 0x0001:0x0000;
            break;
        case 011b:  // <=
            sum = (a<=b)? 0x0001:0x0000;
            break;
        case 100b:  // >
            sum = (a>b)? 0x0001:0x0000;
            break;
        case 101b:  // >=
            sum = (a>=b)? 0x0001:0x0000;
            break;
        case 110b:  // !=
            sum = (a!=b)? 0x0001:0x0000;
            break;
        case 111b:  // ==
            sum = (a==b)? 0x0001:0x0000;
            break;
        case 1000b:     // &
            sum = a&b;
            break;
    }
    return sum;
}

bit[31:0] add(bit[31:0] a, bit[31:0] b, bit[2:0] ctrl)
{
    note add.library = "1";
    note add.si = "data";
    note add.amount = "data";
    note add.so = "data";
    note add.ctrl = "control";

    note add.type = "adda";
    note add.width = "272";
    note add.height = "420";
    note add.cost = "60";
    note add.pipelined = "0";
    note add.delay = "1";
    note add.bits = "32";
    note add.operation = "+";
    note add.num_wports= "2";
    note add.num_rports = "1";

    bit[31:0] so;
    so = a+b;
    return so;
}

void RF(event clk, bit[0:0] rst, bit[31:0] inp,
    bit[1:0] raA, bit[1:0] raB, bit[0:0] reA, bit[0:0] reB,
    bit[1:0] wa, bit[0:0] we, bit[31:0] outA, bit[31:0] outB)
{
    note RF.library = "1";
    note RF.type = "RF";
    note RF.size = "8";
    note RF.width = "272";
    note RF.height = "420";
    note RF.cost = "60";
    note RF.pipelined = "0";
    note RF.delay = "0";
    note RF.num_inports= "1";
    note RF.num_outports = "2";
    note RF.bits = "32";
}

void PC(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note PC.library = "1";
    note PC.type = "reg";
    note PC.size = "1";
    note PC.width = "100";
    note PC.height = "220";
    note PC.cost = "30";
    note PC.pipelined = "0";
    note PC.delay = "0";
    note PC.num_inports= "1";
    note PC.num_outports = "1";
    note PC.bits = "32";
}


void Status(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note Status.library = "1";
    note Status.type = "reg";
    note Status.size = "1";
    note Status.width = "100";
    note Status.height = "220";
    note Status.cost = "30";
    note Status.pipelined = "0";
    note Status.delay = "0";
    note Status.num_inports= "1";
    note Status.num_outports = "1";
    note Status.bits = "32";
}


void IR(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note IR.library = "1";
    note IR.type = "reg";
    note IR.size = "1";
    note IR.width = "100";
    note IR.height = "220";
    note IR.cost = "30";
    note IR.pipelined = "0";
    note IR.delay = "0";
    note IR.num_inports= "1";
    note IR.num_outports = "1";
    note IR.bits = "32";
}

void AR(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note AR.library = "1";
    note AR.type = "reg";
    note AR.size = "1";
    note AR.width = "100";
    note AR.height = "220";
    note AR.cost = "30";
    note AR.pipelined = "0";
    note AR.delay = "0";
    note AR.num_inports= "1";
    note AR.num_outports = "1";
    note AR.bits = "32";
}
```

```
void DR(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note DR.library = "1";
    note DR.type = "reg";
    note DR.size = "1";
    note DR.width = "100";
    note DR.height = "220";
    note DR.cost = "30";
    note DR.pipelined = "0";
    note DR.delay = "0";
    note DR.num_inports= "1";
    note DR.num_outports = "1";
    note DR.bits = "32";
}

void MEM(event clk, bit[0:0] rst, bit[31:0] inp,
    bit[1:0] raA, bit[1:0] raB, bit[0:0] reA, bit[0:0] reB,
    bit[1:0] wa, bit[0:0] we, bit[31:0] outA, bit[31:0] outB)
{
    note MEM.library = "1";
    note MEM.type = "mem";
    note MEM.size = "65536";
    note MEM.width = "272";
    note MEM.height = "420";
    note MEM.cost = "60";
    note MEM.pipelined = "0";
    note MEM.delay = "0";
    note MEM.num_rwports= "1";
    note MEM.num_aports = "1";
    note MEM.bits = "32";
}

void bus(bit[31:0] outp, bit[31:0] inp)
{
    note bus.library = "1";
    note bus.type = "bus";
    note bus.width = "1";
    note bus.height = "1";
    note bus.cost = "60";
    note bus.delay = "0";
    note bus.bits = "32";
}

note alu.num = "1";
note add.num = "1";
note RF.num = "1";
note MEM.num = "1";
note PC.num = "1";
note AR.num = "1";
note DR.num = "1";
note IR.num = "1";
note Status.num = "1";
note bus.num = "5";
```

B.2 Design 1 output: datapath with special registers

```
/*********************************************************
* SpecC code generated by 'genc'
* Date: Mon Apr  1 15:16:15 2002
* User: haoboy
*********************************************************/
import "lib";
behavior ISS(in event clk, in bit[0:0] rst, in bit[15:0] InAddr,
            out bit[15:0] OutAddr, in bit[0:0] start, out bit[0:0] done)
{

    note ISS.scheduled = "1";
    note ISS.fubind = "1";
    note ISS.regbind = "1";
    note ISS.busbind = "1";
    note ISS.InAddr = "data";
    note ISS.clk = "clk";
    note ISS.done = "ctrl";
    note ISS.rst = "rst";
    note ISS.start = "ctrl";

    bit[31:0] add0(bit[31:0] a, bit[31:0] b, bit[2:0] ctrl)
    {
        return add(a, b, ctrl);
    }

    bit[31:0] alu0(bit[31:0] a, bit[31:0] b, bit[2:0] ctrl)
    {
        return alu(a, b, ctrl);
    }

    void main(void)
    {
        bit[15:0] AR;
        bit[31:0] AR0;
        bit[15:0] DR;
        bit[31:0] DR0;
        bit[15:0] IR;
        bit[31:0] IR0;
        bit[15:0] MEM[65536];
        bit[31:0] MEM0[65536];
        bit[15:0] PC;
        bit[31:0] PC0;
        bit[15:0] RF[8];
        bit[31:0] RF0[4];
        bit[15:0] Status;
        bit[31:0] Status0;
        bit[0:0] _ctrl_;
        bit[31:0] bus0;
        bit[31:0] bus1;
        bit[31:0] bus2;
        bit[31:0] bus3;
```

```
bit[31:0] bus4;
enum state { S0, F0, R0, M0, MIm0, MIm1, MDi0, MDi1, MDi2, MRe0,
MRe1, MRe2, MIn0, MIn1, MIn2, MEr0, B0, BJ0, BB0, BB1, BB2, BS0,
BS1, BR0, BR1, BEr0, I0, I1, I2, I3, I4, IEr0, X0, X1, X2, X3,
X4, X5, X6, X7, X8, X9, X10, X11 } state;
while (1)
{
    wait(clk);
    if (rst)
    {
        state = S0;
    }
    switch (state)
    {
        case S0 :
        {
            done = 0;
            PC0 = InAddr;
            if (start!=0)
            {
                state = F0;
            }
            else
            {
                state = S0;
            }
            break;
        }
        case F0 :
        {
            bus1 = PC0;
            bus0 = add0(bus1, 1, 0);
            PC0 = bus0;
            bus2 = MEM0[bus1];
            IR0 = bus2;
            switch (IR0[15:14])
            {
                case 0 :
                {
                    state = R0;
                    break;
                }
                case 1 :
                {
                    state = M0;
                    break;
                }
                case 2 :
                {
                    state = B0;
                    break;
                }
                case 3 :
```

```
                {
                    state = I0;
                    break;
                }
            }
            break;
        }
        case R0 :
        {
            bus1 = RF0[IR0[5:3]];
            bus0 = RF0[IR0[2:0]];
            bus2 = add0(bus1, bus0, 0);
            RF0[IR0[8:6]] = bus2;
            state = F0;
            break;
        }
        case M0 :
        {
            switch (IR0[12:9])
            {
                case 0 :
                {
                    state = MIm0;
                    break;
                }
                case 1 :
                {
                    state = MDi0;
                    break;
                }
                case 2 :
                {
                    state = MRe0;
                    break;
                }
                case 3 :
                {
                    state = MIn0;
                    break;
                }
            }
            break;
        }
        case MIm0 :
        {
            bus0 = IR0[13:13];
            if (alu0(bus0, 0, 7))
            {
                state = MIm1;
            }
            else
            {
                state = MEr0;
            }
            break;
        }
        case MIm1 :
        {
            bus1 = PC0;
            bus0 = add0(bus1, 1, 0);
            PC0 = bus0;
            bus2 = MEM0[bus1];
            RF0[IR0[8:6]] = bus2;
            state = F0;
            break;
        }
        case MDi0 :
        {
            bus0 = IR0[13:13];
            if (alu0(bus0, 0, 7))
            {
                state = MDi1;
            }
            else
            {
                state = MDi2;
            }
            break;
        }
        case MDi1 :
        {
            bus1 = PC0;
            bus0 = add0(bus1, 1, 0);
            PC0 = bus0;
            bus2 = MEM0[bus1];
            AR0 = bus2;
            state = X0;
            break;
        }
        case X0 :
        {
            bus1 = AR0;
            bus2 = MEM0[bus1];
            RF0[IR0[8:6]] = bus2;
            state = F0;
            break;
        }
        case MDi2 :
        {
            bus1 = PC0;
            bus0 = add0(bus1, 1, 0);
            PC0 = bus0;
            bus2 = RF0[IR0[5:3]];
            DR0 = bus2;
            bus3 = MEM0[bus1];
            AR0 = bus3;
            state = X1;
            break;
        }
        case X1 :
        {
            bus1 = AR0;
            bus2 = DR0;
            MEM0[bus1] = bus2;
            state = F0;
            break;
        }
        case MRe0 :
        {
            bus0 = IR0[13:13];
            if (alu0(bus0, 0, 7))
            {
                state = MRe1;
            }
            else
            {
                state = MRe2;
            }
            break;
        }
        case MRe1 :
        {
            bus1 = RF0[IR0[8:6]];
            bus0 = RF0[IR0[2:0]];
            bus2 = add0(bus1, bus0, 0);
            AR0 = bus2;
            bus3 = PC0;
            bus4 = MEM0[bus3];
            RF0[IR0[8:6]] = bus4;
            state = X2;
            break;
        }
        case X2 :
        {
            bus1 = PC0;
            bus0 = add0(bus1, 1, 0);
            PC0 = bus0;
            bus3 = AR0;
            bus2 = MEM0[bus3];
            RF0[IR0[8:6]] = bus2;
            state = F0;
            break;
        }
        case MRe2 :
        {
            bus0 = RF0[IR0[5:3]];
            DR0 = bus0;
            state = X3;
            break;
        }
        case X3 :
        {
            bus1 = RF0[IR0[8:6]];
            bus0 = RF0[IR0[2:0]];
            bus2 = add0(bus1, bus0, 0);
            AR0 = bus2;
            bus3 = PC0;
            bus4 = MEM0[bus3];
            RF0[IR0[8:6]] = bus4;
            state = X4;
            break;
        }
        case X4 :
        {
            bus1 = PC0;
            bus0 = add0(bus1, 1, 0);
            PC0 = bus0;
            bus3 = AR0;
            bus2 = DR0;
            MEM0[bus3] = bus2;
            state = F0;
            break;
        }
        case MIn0 :
        {
            bus0 = IR0[13:13];
            if (alu0(bus0, 0, 7))
            {
                state = MIn1;
            }
            else
            {
                state = MIn2;
            }
            break;
        }
        case MIn1 :
        {
            bus1 = PC0;
            bus0 = add0(bus1, 1, 0);
            PC0 = bus0;
            bus2 = MEM0[bus1];
            AR0 = bus2;
            state = X5;
            break;
        }
        case X5 :
        {
            bus1 = AR0;
            bus2 = MEM0[bus1];
            AR0 = bus2;
            state = X6;
            break;
        }
        case X6 :
        {
            bus1 = AR0;
            bus2 = MEM0[bus1];
            RF0[IR0[8:6]] = bus2;
            state = F0;
            break;
        }
        case MIn2 :
        {
            bus1 = PC0;
            bus0 = add0(bus1, 1, 0);
            PC0 = bus0;
            bus2 = RF0[IR0[5:3]];
            DR0 = bus2;
            bus3 = MEM0[bus1];
            AR0 = bus3;
            state = X7;
            break;
        }
        case X7 :
        {
            bus1 = AR0;
            bus2 = MEM0[bus1];
            AR0 = bus2;
            state = X8;
            break;
        }
        case X8 :
        {
            bus1 = AR0;
            bus2 = DR0;
            MEM0[bus1] = bus2;
            state = F0;
            break;
        }
        case MEr0 :
        {
            state = S0;
            break;
        }
        case B0 :
        {
            switch (IR0[13:9])
            {
                case 0 :
                {
                    state = BJ0;
                    break;
                }
                case 1 :
```

```
                {
                    state = BB0;
                    break;
                }
                case 2 :
                {
                    state = BS0;
                    break;
                }
                case 3 :
                {
                    state = BR0;
                    break;
                }
                default :
                {
                    state = BEr0;
                    break;
                }
            }
            break;
        }
        case BJ0 :
        {
            bus1 = PC0;
            bus2 = MEM0[bus1];
            PC0 = bus2;
            state = F0;
            break;
        }
        case BB0 :
        {
            bus0 = Status0;
            if (alu0(bus0, 0, 7))
            {
                state = BB1;
            }
            else
            {
                state = BB2;
            }
            break;
        }
        case BB1 :
        {
            bus1 = PC0;
            bus0 = add0(bus1, 1, 0);
            PC0 = bus0;
            state = F0;
            break;
        }
        case BB2 :
        {
            bus1 = PC0;
            bus2 = MEM0[bus1];
            PC0 = bus2;
            state = F0;
            break;
        }
        case BS0 :
        {
            bus1 = RF0[IR0[5:3]];
            bus2 = add0(bus1, 1, 0);
            RF0[IR0[5:3]] = bus2;
            bus3 = PC0;
            bus4 = MEM0[bus3];
            DR0 = bus4;
            bus0 = RF0[IR0[5:3]];
            AR0 = bus0;
            state = X9;
            break;
        }
        case X9 :
        {
            bus1 = PC0;
            bus0 = add0(bus1, 1, 0);
            PC0 = bus0;
            state = X10;
            break;
        }
        case X10 :
        {
            bus0 = DR0;
            PC0 = bus0;
            bus1 = AR0;
            bus2 = PC0;
            MEM0[bus1] = bus2;
            state = F0;
            break;
        }
        case BR0 :
        {
            bus1 = RF0[IR0[5:3]];
            bus2 = add0(bus1, 1, 0);
            RF0[IR0[5:3]] = bus2;
            bus0 = RF0[0];
            RF0[0] = bus0;
            bus3 = AR0;
            bus4 = MEM0[bus3];
            AR0 = bus4;
            state = X11;
            break;
        }
        case X11 :
        {
            bus1 = AR0;
```

```
        bus2 = MEM0[bus1];
        PC0 = bus2;
        state = F0;
        break;
    }
    case BEr0 :
    {
        state = S0;
        break;
    }
    case I0 :
    {
        switch (IR0[13:9])
        {
            case 0 :
            {
                state = F0;
                break;
            }
            case 1 :
            {
                state = I1;
                break;
            }
            case 2 :
            {
                state = I2;
                break;
            }
            case 3 :
            {
                state = I3;
                break;
            }
            case 4 :
            {
                state = I4;
                break;
            }
            default :
            {
                state = IEr0;
                break;
            }
        }
        break;
    }
    case I1 :
    {
        RF0[IR0[8:6]] = 0;
        state = F0;
        break;
    }
```

```
                case I2 :
                {
                    bus0 = RF0[IR0[5:3]];
                    bus1 = RF0[IR0[2:0]];
                    bus2 = alu0(bus0, bus1, 1);
                    Status0 = bus2;
                    state = F0;
                    break;
                }
                case I3 :
                {
                    Status0[IR0[8:6]] = 1;
                    state = F0;
                    break;
                }
                case I4 :
                {
                    Status0[IR0[8:6]] = 0;
                    state = F0;
                    break;
                }
                case IEr0 :
                {
                    state = S0;
                    break;
                }
            }
        }
    }

};
```

C. Special Note

We only have output code for design 1. Since we are still working on the syntax of the output code, we do not put the output code for design 2-5 in this version. We will append the output code soon after we make a decision about the syntax of Pipeline/Multicyle operation.

D. Design 2:Datapath with register file only

D.1 Design 2 input: RTL component library

```
bit[31:0] alu(bit[31:0] a, bit[31:0] b, bit[2:0] ctrl)
{
    note  alu.library = "1";
    note  alu.a="data";
    note  alu.b="data";
    note  alu.sum="data";
    note  alu.ctrl="control";

    note  alu.type="rca";
    note  alu.width="472";
    note  alu.height="920";
    note  alu.cost="100";
    note  alu.pipelined = "0";
    note  alu.delay="1";
    note  alu.bits="32";
    note  alu.operation="+,-,<,<=,>,>=,!=,==,&,+:,-:,+=,-=";
    note  alu.num_wports= "2";
    note  alu.num_rports = "1";
    bit[31:0] sum;

    switch(ctrl)   {
        case 000b:  // +
            sum = a+b;
            break;
        case 001b:  // -
            sum = a-b;
            break;
        case 010b:  // <
            sum = (a<b)? 0x0001:0x0000;
            break;
        case 011b:  // <=
            sum = (a<=b)? 0x0001:0x0000;
            break;
        case 100b:  // >
            sum = (a>b)? 0x0001:0x0000;
            break;
        case 101b:  // >=
            sum = (a>=b)? 0x0001:0x0000;
            break;
        case 110b:  // !=
            sum = (a!=b)? 0x0001:0x0000;
            break;
        case 111b:  // ==
            sum = (a==b)? 0x0001:0x0000;
            break;
        case 1000b:     // &
            sum = a&b;
            break;
    }
    return sum;
}

bit[31:0] add(bit[31:0] a, bit[31:0] b, bit[2:0] ctrl)
{
    note add.library = "1";
    note add.si = "data";
    note add.amount = "data";
    note add.so = "data";
    note add.ctrl = "control";

    note add.type = "adda";
    note add.width = "272";
    note add.height = "420";
    note add.cost = "60";
    note add.pipelined = "0";
    note add.delay = "1";
    note add.bits = "32";
    note add.operation = "+";
    note add.num_wports= "2";
    note add.num_rports = "1";

    bit[31:0] so;
    so = a+b;
    return so;
}

void RF(event clk, bit[0:0] rst, bit[31:0] inp,
    bit[1:0] raA, bit[1:0] raB, bit[0:0] reA, bit[0:0] reB,
    bit[1:0] wa, bit[0:0] we, bit[31:0] outA, bit[31:0] outB)
{
    note RF.library = "1";
    note RF.type = "RF";
    note RF.size = "16";
    note RF.width = "272";
    note RF.height = "420";
    note RF.cost = "60";
    note RF.pipelined = "0";
    note RF.delay = "0";
    note RF.readReg1="a";
    note RF.readReg2="b";
    note RF.num_inports= "1";
    note RF.num_outports = "2";
    note RF.bits = "32";
}



void Status(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note Status.library = "1";
    note Status.type = "reg";
    note Status.size = "1";
    note Status.width = "100";
    note Status.height = "220";
    note Status.cost = "30";
    note Status.pipelined = "0";
    note Status.delay = "0";
    note Status.num_inports= "1";
    note Status.num_outports = "1";
    note Status.bits = "32";
}


void IR(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note IR.library = "1";
    note IR.type = "reg";
    note IR.size = "1";
    note IR.width = "100";
    note IR.height = "220";
    note IR.cost = "30";
    note IR.pipelined = "0";
    note IR.delay = "0";
    note IR.num_inports= "1";
    note IR.num_outports = "1";
    note IR.bits = "32";
}


void MEM(event clk, bit[0:0] rst, bit[31:0] inp,
    bit[1:0] raA, bit[1:0] raB, bit[0:0] reA, bit[0:0] reB,
    bit[1:0] wa, bit[0:0] we, bit[31:0] outA, bit[31:0] outB)
{
    note MEM.library = "1";
    note MEM.type = "mem";
    note MEM.size = "65536";
    note MEM.width = "272";
    note MEM.height = "420";
    note MEM.cost = "60";
    note MEM.pipelined = "0";
    note MEM.delay = "0";
    note MEM.num_rwports= "1";
    note MEM.num_aports = "1";
    note MEM.bits = "32";
}

void bus(bit[31:0] outp, bit[31:0] inp)
{
    note bus.library = "1";
    note bus.type = "bus";
    note bus.width = "1";
    note bus.height = "1";
    note bus.cost = "60";
    note bus.delay = "0";
    note bus.bits = "32";
}

note alu.num = "1";
note add.num = "1";
note RF.num = "1";
note MEM.num = "1";
note IR.num = "1";
note Status.num = "1";
note bus.num = "3";
```

E. Design 3: Datapath with latched register file

E.1 Design 3 input: RTL component library

```
bit[31:0] alu(bit[31:0] a, bit[31:0] b, bit[2:0] ctrl)
{
    note  alu.library = "1";
    note  alu.a="data";
    note  alu.b="data";
    note  alu.sum="data";
    note  alu.ctrl="control";

    note  alu.type="rca";
    note  alu.width="472";
    note  alu.height="920";
    note  alu.cost="100";
    note  alu.pipelined = "0";
    note  alu.delay="1";
    note  alu.bits="32";
    note  alu.operation="+,-,<,<=,>,>=,!=,==,&,+:,-:,+=,-=";
    note  alu.readReg1="a";
    note  alu.readReg2="b";
    note  alu.num_wports= "2";
    note  alu.num_rports = "1";
    bit[31:0] sum;

    switch(ctrl)   {
        case 000b:  // +
            sum = a+b;
            break;
        case 001b:  // -
            sum = a-b;
            break;
        case 010b:  // <
            sum = (a<b)? 0x0001:0x0000;
            break;
        case 011b:  // <=
            sum = (a<=b)? 0x0001:0x0000;
            break;
        case 100b:  // >
            sum = (a>b)? 0x0001:0x0000;
            break;
        case 101b:  // >=
            sum = (a>=b)? 0x0001:0x0000;
            break;
        case 110b:  // !=
            sum = (a!=b)? 0x0001:0x0000;
            break;
        case 111b:  // ==
            sum = (a==b)? 0x0001:0x0000;
            break;
        case 1000b:     // &
            sum = a&b;
```

```
                break;
        }
        return sum;
}

bit[31:0] add(bit[31:0] a, bit[31:0] b, bit[2:0] ctrl)
{
        note add.library = "1";
        note add.si = "data";
        note add.amount = "data";
        note add.so = "data";
        note add.ctrl = "control";

        note add.type = "adda";
        note add.width = "272";
        note add.height = "420";
        note add.cost = "60";
        note add.pipelined = "0";
        note add.delay = "1";
        note add.bits = "32";
        note add.operation = "+";
        note add.num_wports= "2";
        note add.num_rports = "1";

        bit[31:0] so;
        so = a+b;
        return so;
}

void RF(event clk, bit[0:0] rst, bit[31:0] inp,
        bit[1:0] raA, bit[1:0] raB, bit[0:0] reA, bit[0:0] reB,
        bit[1:0] wa, bit[0:0] we, bit[31:0] outA, bit[31:0] outB)
{
        note RF.library = "1";
        note RF.type = "RF";
        note RF.size = "8";
        note RF.width = "272";
        note RF.height = "420";
        note RF.cost = "60";
        note RF.pipelined = "1";
        note RF.delay = "1";
        note RF.readReg1="a";
        note RF.readReg2="b";
        note RF.num_inports= "1";
        note RF.num_outports = "2";
        note RF.bits = "32";
}

void PC(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
        note PC.library = "1";
        note PC.type = "reg";
        note PC.size = "1";
```

```
    note PC.width = "100";
    note PC.height = "220";
    note PC.cost = "30";
    note PC.pipelined = "0";
    note PC.delay = "0";
    note PC.num_inports= "1";
    note PC.num_outports = "1";
    note PC.bits = "32";
}


void Status(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note Status.library = "1";
    note Status.type = "reg";
    note Status.size = "1";
    note Status.width = "100";
    note Status.height = "220";
    note Status.cost = "30";
    note Status.pipelined = "0";
    note Status.delay = "0";
    note Status.num_inports= "1";
    note Status.num_outports = "1";
    note Status.bits = "32";
}


void IR(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note IR.library = "1";
    note IR.type = "reg";
    note IR.size = "1";
    note IR.width = "100";
    note IR.height = "220";
    note IR.cost = "30";
    note IR.pipelined = "0";
    note IR.delay = "0";
    note IR.num_inports= "1";
    note IR.num_outports = "1";
    note IR.bits = "32";
}

void AR(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note AR.library = "1";
    note AR.type = "reg";
    note AR.size = "1";
    note AR.width = "100";
    note AR.height = "220";
    note AR.cost = "30";
    note AR.pipelined = "0";
    note AR.delay = "0";
    note AR.num_inports= "1";
    note AR.num_outports = "1";
    note AR.bits = "32";
}

void DR(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note DR.library = "1";
    note DR.type = "reg";
    note DR.size = "1";
    note DR.width = "100";
    note DR.height = "220";
    note DR.cost = "30";
    note DR.pipelined = "0";
    note DR.delay = "0";
    note DR.num_inports= "1";
    note DR.num_outports = "1";
    note DR.bits = "32";
}

void MEM(event clk, bit[0:0] rst, bit[31:0] inp,
    bit[1:0] raA, bit[1:0] raB, bit[0:0] reA, bit[0:0] reB,
    bit[1:0] wa, bit[0:0] we, bit[31:0] outA, bit[31:0] outB)
{
    note MEM.library = "1";
    note MEM.type = "mem";
    note MEM.size = "65536";
    note MEM.width = "272";
    note MEM.height = "420";
    note MEM.cost = "60";
    note MEM.pipelined = "0";
    note MEM.delay = "0";
    note MEM.num_rwports= "1";
    note MEM.num_aports = "1";
    note MEM.bits = "32";
}

void bus(bit[31:0] outp, bit[31:0] inp)
{
    note bus.library = "1";
    note bus.type = "bus";
    note bus.width = "1";
    note bus.height = "1";
    note bus.cost = "60";
    note bus.delay = "0";
    note bus.bits = "32";
}

note alu.num = "1";
note add.num = "1";
note RF.num = "1";
note MEM.num = "1";
note PC.num = "1";
note AR.num = "1";
note DR.num = "1";
note IR.num = "1";
note Status.num = "1";
note bus.num = "5";
```

F. Design 4: Datapath with pipelined functional units

F.1 Design 4 input: RTL component library

```
bit[31:0] alu(bit[31:0] a, bit[31:0] b, bit[2:0] ctrl)
{
    note  alu.library = "1";
    note  alu.a="data";
    note  alu.b="data";
    note  alu.sum="data";
    note  alu.ctrl="control";

    note  alu.type="rca";
    note  alu.width="472";
    note  alu.height="920";
    note  alu.cost="100";
    note  alu.pipelined = "1";
    note  alu.delay="2";
    note  alu.bits="32";
    note  alu.operation="+,-,<,<=,>,>=,!=,==,&,+:,-:,+=,-=";
    note  alu.num_wports= "2";
    note  alu.num_rports = "1";
    bit[31:0] sum;

    switch(ctrl)   {
        case 000b:  // +
            sum = a+b;
            break;
        case 001b:  // -
            sum = a-b;
            break;
        case 010b:  // <
            sum = (a<b)? 0x0001:0x0000;
            break;
        case 011b:  // <=
            sum = (a<=b)? 0x0001:0x0000;
            break;
        case 100b:  // >
            sum = (a>b)? 0x0001:0x0000;
            break;
        case 101b:  // >=
            sum = (a>=b)? 0x0001:0x0000;
            break;
        case 110b:  // !=
            sum = (a!=b)? 0x0001:0x0000;
            break;
        case 111b:  // ==
            sum = (a==b)? 0x0001:0x0000;
            break;
        case 1000b:     // &
            sum = a&b;
            break;
    }
    return sum;
}

bit[31:0] add(bit[31:0] a, bit[31:0] b, bit[2:0] ctrl)
{
    note add.library = "1";
    note add.si = "data";
    note add.amount = "data";
    note add.so = "data";
    note add.ctrl = "control";

    note add.type = "adda";
    note add.width = "272";
    note add.height = "420";
    note add.cost = "60";
    note add.pipelined = "1";
    note add.delay = "2";
    note add.bits = "32";
    note add.operation = "+";
    note add.num_wports= "2";
    note add.num_rports = "1";

    bit[31:0] so;
    so = a+b;
    return so;
}

void RF(event clk, bit[0:0] rst, bit[31:0] inp,
    bit[1:0] raA, bit[1:0] raB, bit[0:0] reA, bit[0:0] reB,
    bit[1:0] wa, bit[0:0] we, bit[31:0] outA, bit[31:0] outB)
{
    note RF.library = "1";
    note RF.type = "RF";
    note RF.size = "8";
    note RF.width = "272";
    note RF.height = "420";
    note RF.cost = "60";
    note RF.pipelined = "0";
    note RF.delay = "0";
    note RF.num_inports= "1";
    note RF.num_outports = "2";
    note RF.bits = "32";
}

void PC(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note PC.library = "1";
    note PC.type = "reg";
    note PC.size = "1";
    note PC.width = "100";
    note PC.height = "220";
    note PC.cost = "30";
    note PC.pipelined = "0";
    note PC.delay = "0";
    note PC.num_inports= "1";
    note PC.num_outports = "1";
    note PC.bits = "32";
}


void Status(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note Status.library = "1";
    note Status.type = "reg";
    note Status.size = "1";
    note Status.width = "100";
    note Status.height = "220";
    note Status.cost = "30";
    note Status.pipelined = "0";
    note Status.delay = "0";
    note Status.num_inports= "1";
    note Status.num_outports = "1";
    note Status.bits = "32";
}


void IR(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note IR.library = "1";
    note IR.type = "reg";
    note IR.size = "1";
    note IR.width = "100";
    note IR.height = "220";
    note IR.cost = "30";
    note IR.pipelined = "0";
    note IR.delay = "0";
    note IR.num_inports= "1";
    note IR.num_outports = "1";
    note IR.bits = "32";
}

void AR(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note AR.library = "1";
    note AR.type = "reg";
    note AR.size = "1";
    note AR.width = "100";
    note AR.height = "220";
    note AR.cost = "30";
    note AR.pipelined = "0";
    note AR.delay = "0";
    note AR.num_inports= "1";
    note AR.num_outports = "1";
    note AR.bits = "32";
}

```

```
void DR(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note DR.library = "1";
    note DR.type = "reg";
    note DR.size = "1";
    note DR.width = "100";
    note DR.height = "220";
    note DR.cost = "30";
    note DR.pipelined = "0";
    note DR.delay = "0";
    note DR.num_inports= "1";
    note DR.num_outports = "1";
    note DR.bits = "32";
}

void MEM(event clk, bit[0:0] rst, bit[31:0] inp,
    bit[1:0] raA, bit[1:0] raB, bit[0:0] reA, bit[0:0] reB,
    bit[1:0] wa, bit[0:0] we, bit[31:0] outA, bit[31:0] outB)
{
    note MEM.library = "1";
    note MEM.type = "mem";
    note MEM.size = "65536";
    note MEM.width = "272";
    note MEM.height = "420";
    note MEM.cost = "60";
    note MEM.pipelined = "0";
    note MEM.delay = "0";
    note MEM.num_rwports= "1";
    note MEM.num_aports = "1";
    note MEM.bits = "32";
}

void bus(bit[31:0] outp, bit[31:0] inp)
{
    note bus.library = "1";
    note bus.type = "bus";
    note bus.width = "1";
    note bus.height = "1";
    note bus.cost = "60";
    note bus.delay = "0";
    note bus.bits = "32";
}

note alu.num = "1";
note add.num = "1";
note RF.num = "1";
note MEM.num = "1";
note PC.num = "1";
note AR.num = "1";
note DR.num = "1";
note IR.num = "1";
note Status.num = "1";
note bus.num = "5";
```

G. Design 5: Datapath with multicycle memory

G.1 Design 5 input: RTL component library

```
bit[31:0] alu(bit[31:0] a, bit[31:0] b, bit[2:0] ctrl)
{
    note  alu.library = "1";
    note  alu.a="data";
    note  alu.b="data";
    note  alu.sum="data";
    note  alu.ctrl="control";

    note  alu.type="rca";
    note  alu.width="472";
    note  alu.height="920";
    note  alu.cost="100";
    note  alu.pipelined = "1";
    note  alu.delay="2";
    note  alu.bits="32";
    note  alu.operation="+,-,<,<=,>,>=,!=,==,&,+:,-:,+=,-=";
    note  alu.num_wports= "2";
    note  alu.num_rports = "1";
    bit[31:0] sum;

    switch(ctrl)   {
        case 000b:  // +
            sum = a+b;
            break;
        case 001b:  // -
            sum = a-b;
            break;
        case 010b:  // <
            sum = (a<b)? 0x0001:0x0000;
            break;
        case 011b:  // <=
            sum = (a<=b)? 0x0001:0x0000;
            break;
        case 100b:  // >
            sum = (a>b)? 0x0001:0x0000;
            break;
        case 101b:  // >=
            sum = (a>=b)? 0x0001:0x0000;
            break;
        case 110b:  // !=
            sum = (a!=b)? 0x0001:0x0000;
            break;
        case 111b:  // ==
            sum = (a==b)? 0x0001:0x0000;
            break;
        case 1000b:     // &
            sum = a&b;
            break;
    }
    return sum;
}

bit[31:0] add(bit[31:0] a, bit[31:0] b, bit[2:0] ctrl)
{
    note add.library = "1";
    note add.si = "data";
    note add.amount = "data";
    note add.so = "data";
    note add.ctrl = "control";

    note add.type = "adda";
    note add.width = "272";
    note add.height = "420";
    note add.cost = "60";
    note add.pipelined = "1";
    note add.delay = "2";
    note add.bits = "32";
    note add.operation = "+";
    note add.num_wports= "2";
    note add.num_rports = "1";

    bit[31:0] so;
    so = a+b;
    return so;
}

void RF(event clk, bit[0:0] rst, bit[31:0] inp,
    bit[1:0] raA, bit[1:0] raB, bit[0:0] reA, bit[0:0] reB,
    bit[1:0] wa, bit[0:0] we, bit[31:0] outA, bit[31:0] outB)
{
    note RF.library = "1";
    note RF.type = "RF";
    note RF.size = "4";
    note RF.width = "272";
    note RF.height = "420";
    note RF.cost = "60";
    note RF.pipelined = "1";
    note RF.delay = "1";
    note RF.readReg1="a";
    note RF.readReg2="b";
    note RF.num_inports= "1";
    note RF.num_outports = "2";
    note RF.bits = "32";
}

void PC(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note PC.library = "1";
    note PC.type = "reg";
    note PC.size = "1";
    note PC.width = "100";
    note PC.height = "220";
    note PC.cost = "30";
    note PC.pipelined = "0";
    note PC.delay = "0";
    note PC.num_inports= "1";
    note PC.num_outports = "1";
    note PC.bits = "32";
}


void Status(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note Status.library = "1";
    note Status.type = "reg";
    note Status.size = "1";
    note Status.width = "100";
    note Status.height = "220";
    note Status.cost = "30";
    note Status.pipelined = "0";
    note Status.delay = "0";
    note Status.num_inports= "1";
    note Status.num_outports = "1";
    note Status.bits = "32";
}


void IR(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note IR.library = "1";
    note IR.type = "reg";
    note IR.size = "1";
    note IR.width = "100";
    note IR.height = "220";
    note IR.cost = "30";
    note IR.pipelined = "0";
    note IR.delay = "0";
    note IR.num_inports= "1";
    note IR.num_outports = "1";
    note IR.bits = "32";
}

void AR(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note AR.library = "1";
    note AR.type = "reg";
    note AR.size = "1";
    note AR.width = "100";
    note AR.height = "220";
    note AR.cost = "30";
    note AR.pipelined = "0";
    note AR.delay = "0";
    note AR.num_inports= "1";
    note AR.num_outports = "1";
    note AR.bits = "32";
```

```
}

void DR(event clk, bit[0:0] rst, bit[31:0] inp, bit[31:0] outp)
{
    note DR.library = "1";
    note DR.type = "reg";
    note DR.size = "1";
    note DR.width = "100";
    note DR.height = "220";
    note DR.cost = "30";
    note DR.pipelined = "0";
    note DR.delay = "0";
    note DR.num_inports= "1";
    note DR.num_outports = "1";
    note DR.bits = "32";
}

void MEM(event clk, bit[0:0] rst, bit[31:0] inp,
    bit[1:0] raA, bit[1:0] raB, bit[0:0] reA, bit[0:0] reB,
    bit[1:0] wa, bit[0:0] we, bit[31:0] outA, bit[31:0] outB)
{
    note MEM.library = "1";
    note MEM.type = "mem";
    note MEM.size = "65536";
    note MEM.width = "272";
    note MEM.height = "420";
    note MEM.cost = "60";
    note MEM.pipelined = "1";
    note MEM.delay = "1";
    note MEM.num_rwports= "1";
    note MEM.num_aports = "1";
    note MEM.bits = "32";
}

void bus(bit[31:0] outp, bit[31:0] inp)
{
    note bus.library = "1";
    note bus.type = "bus";
    note bus.width = "1";
    note bus.height = "1";
    note bus.cost = "60";
    note bus.delay = "0";
    note bus.bits = "32";
}

note alu.num = "1";
note add.num = "1";
note RF.num = "1";
note MEM.num = "1";
note PC.num = "1";
note AR.num = "1";
note DR.num = "1";
note IR.num = "1";
note Status.num = "1";
note bus.num = "5";
```