Principles Of
Digital Design

Processor Design

Instruction set

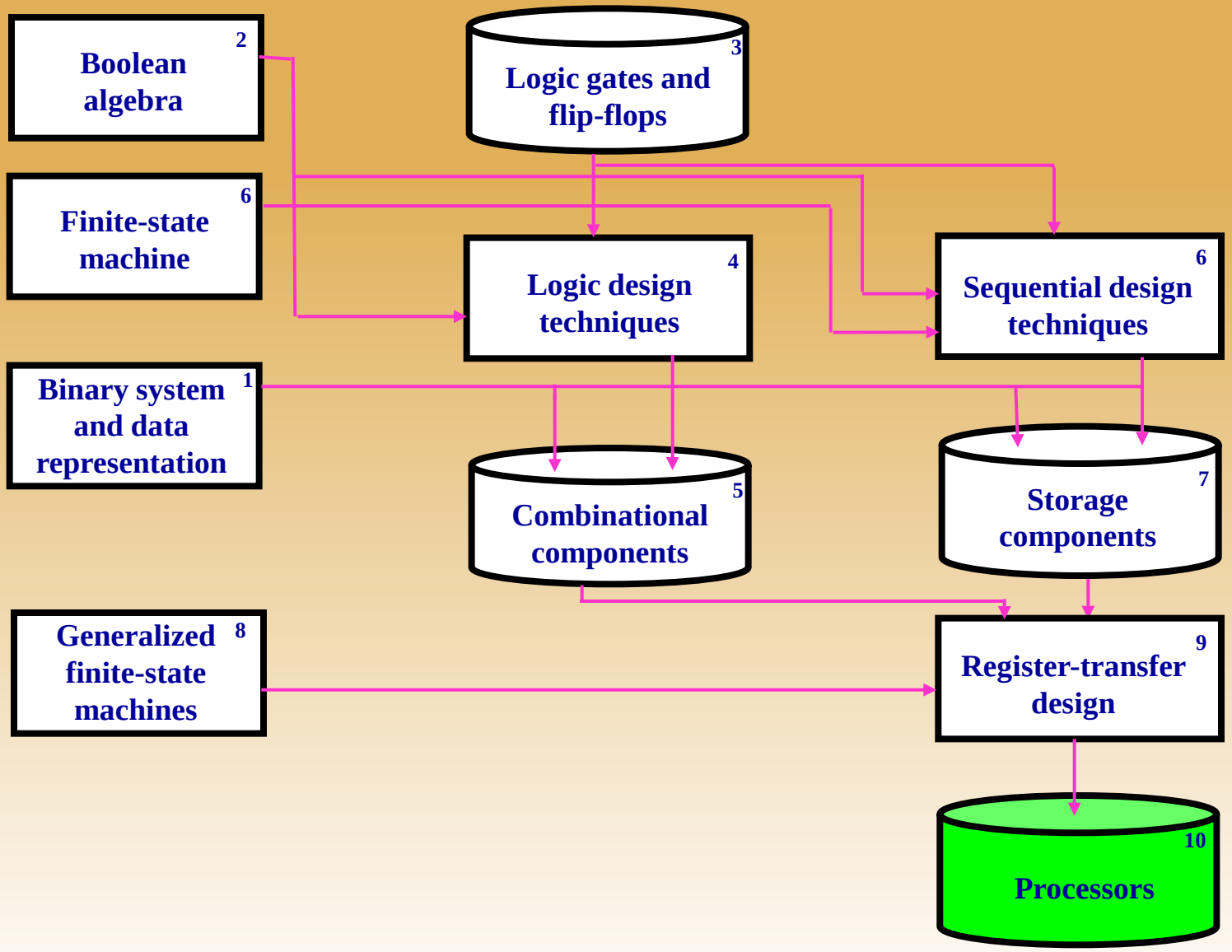
Addressing modes

Processor architecture

Instruction pipelining

Forwarding and branch prediction

Status



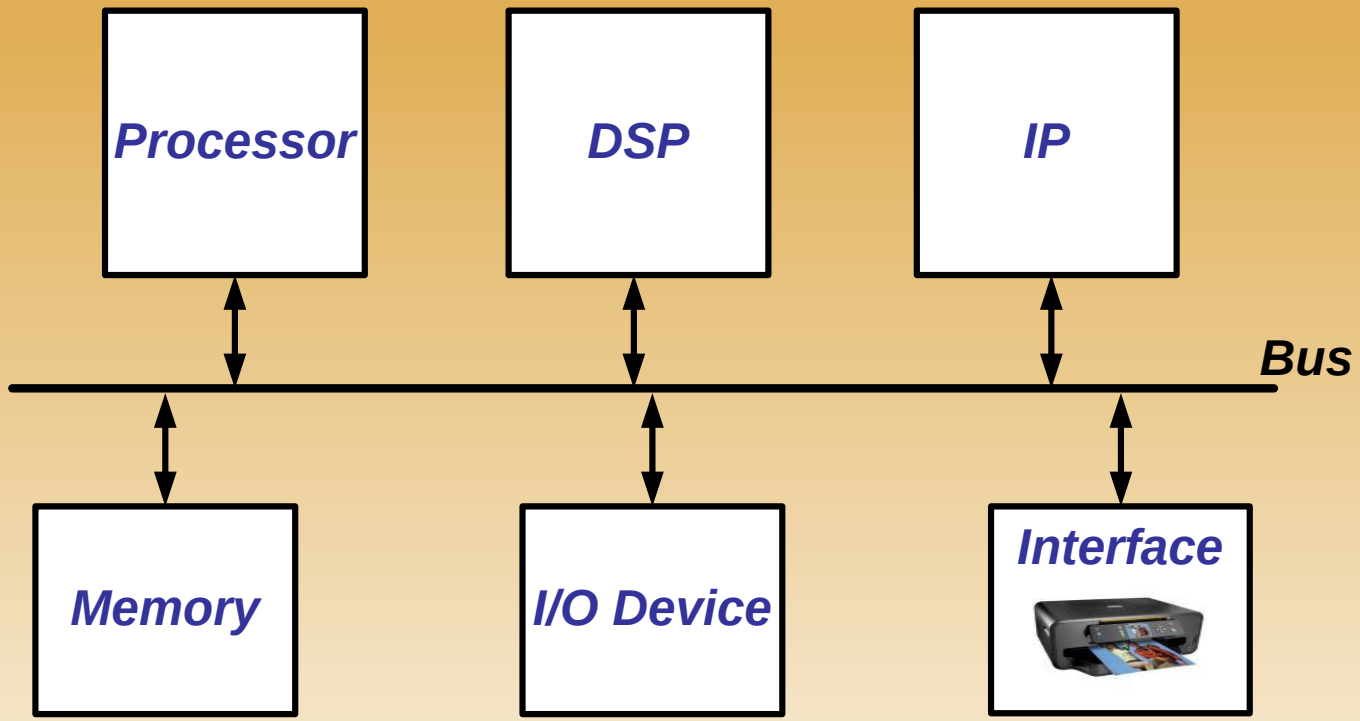
Basic definitions

- Processor controls overall system operations supervising keyboards, monitors, disks, tapes and other Input/Output devices.

- ASIC (Application Specification Integrated Circuit) is a co-processor (such as DSP) that executes one or more specific tasks much faster than the processor itself. For this reason, such ASICs are called accelerators, since the processor offloads computationally intensive tasks to them.

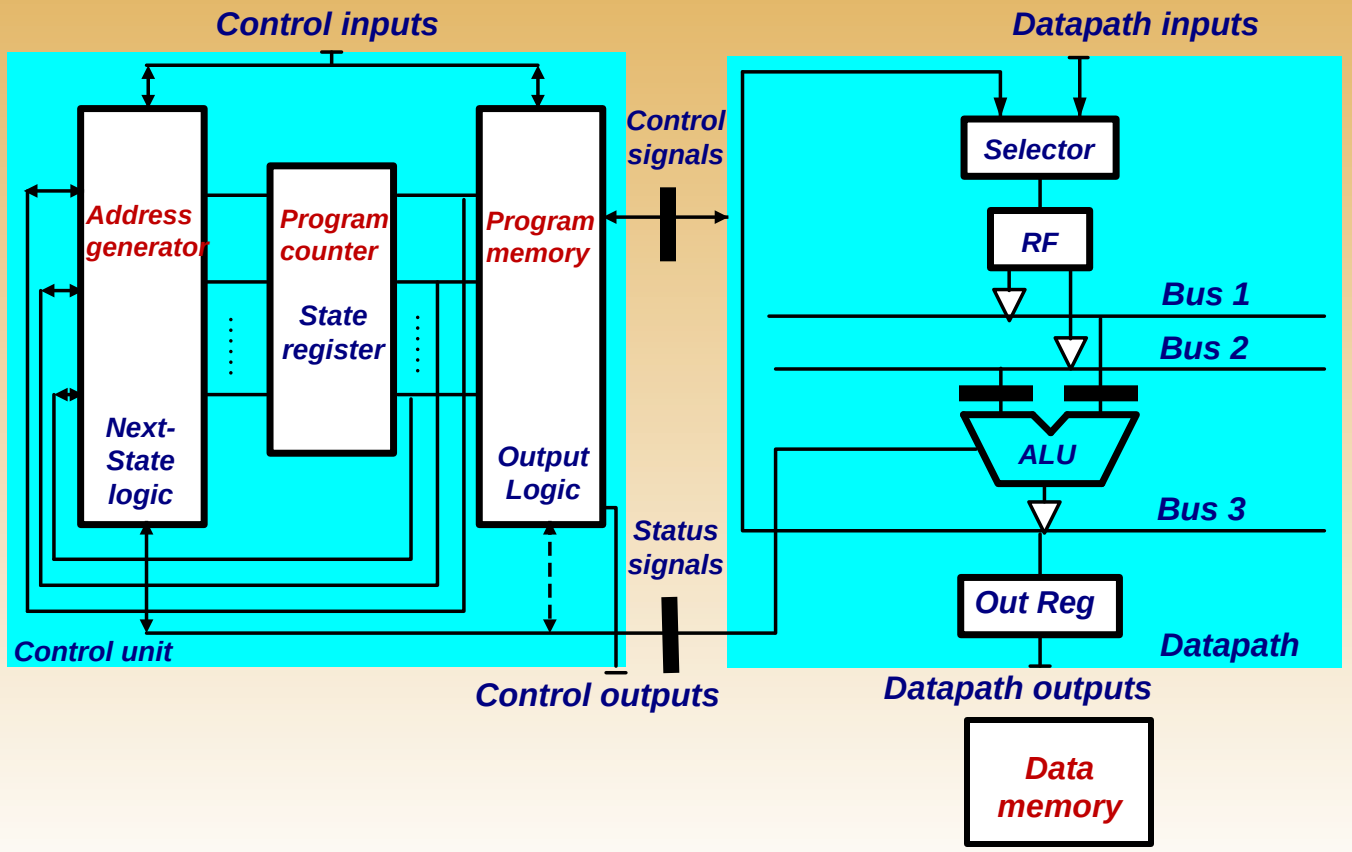
- IP (Intellectual Property component) is a custom hardware specially designed for some specific tasks (such as video processing) that will execute much faster than on an ordinary processor or co-processor.

Basic computer architecture



Old Controler/Datapath diagram

- Four stage control pipeline
- State register, output logic, next-state logic
- Two stage datapath pipeline
- No memory



Instruction set

An instruction is a encoded control word grouped into a number of different fields, such as

- **Operation code**
- **Instruction type**
- **Addressing mode**
- **Constant field**

A typical instruction, such as, a = b + c, where a, b, and c are stored in memory at location A, B, and C can be expressed in assembly language format,

Add A, B, C

or, register-transfer format

Mem[A] <- Mem[B]+ Mem[C]

Typical instruction description

- **Register-to-register instructions**

  Add RA, RB, RC (RF[A] <- RF [B]+ RF [C])

- **Memory instructions (load and store)**

  Load R2, A (RF[2] <- MEM [A])

  Store A, R2 (MEM[A] <- RF [2])

- **Control instructions (branch instruction)**

  Beq R2, R3, A

  if R2=R3 then PC <- MEM[A]

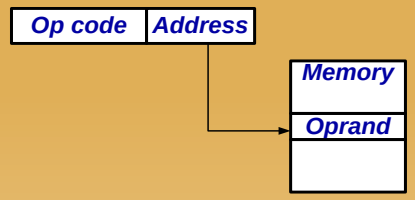
  if not R2=R3 then PC <- PC+1

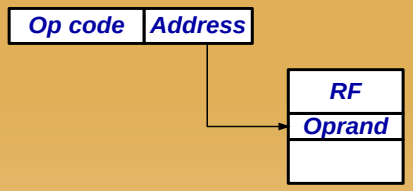
Addressing modes

Implied

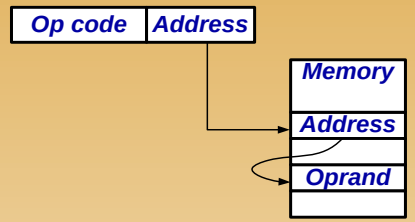
Immediate

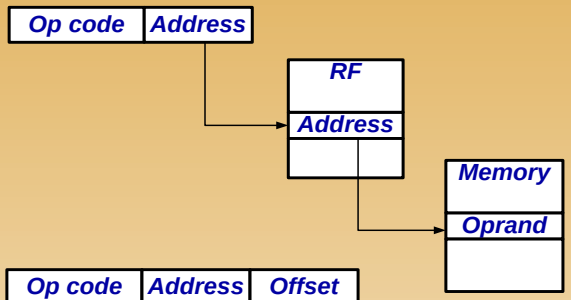
Direct



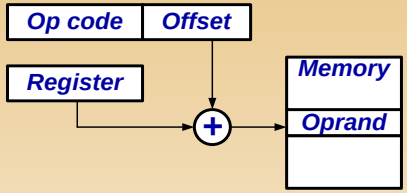


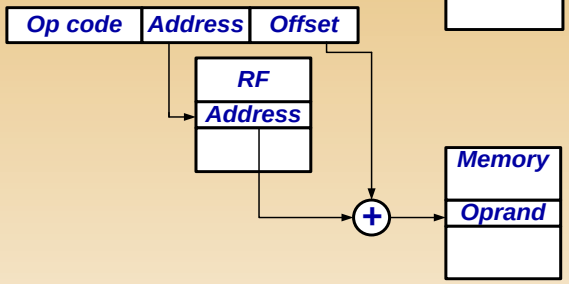
Indirect



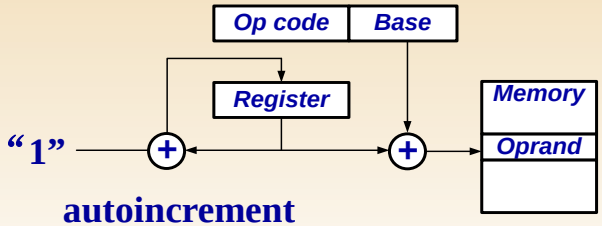


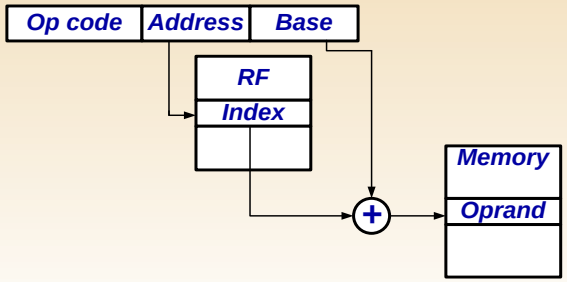
Relative





Indexed





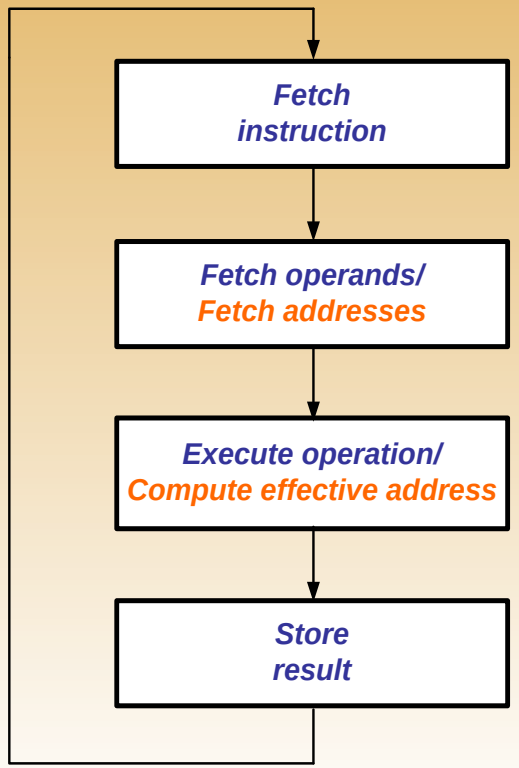
Instruction-set design

Program size vs. processor size

- Complex Instruction-set
  - powerful instructions -> shorter programs
  - powerful instructions -> complex datapath, control unit
  - complex instructions -> several clock cycles
  - complex datapath, control unit -> longer clock period
  - complex instructions -> poor pipelining
- Reduced Instruction-set
  - simple instructions -> longer programs
  - simple instructions -> simple datapath, control unit
  - simple instructions -> single clock cycle
  - simple datapath -> shorter clock period
  - simple instructions -> excellent pipeline

Reduced instruction-set cycle

- Register (add, shift,…) and misc. (set, clear,…) instructions do not need address fetch and effective address computation
- Memory (load, store,…) and control instructions (jump, branch,…) do not need operand fetch and operation execution
- Thus, we can share operand fetches and address fetches
- We can also share operand execution and effective address computation
- Therefore, instruction cycle is reduced to 4 steps



Instruction-set for a 32-bit processor

(a) Register instructions:

Arithmetic,
Logic,
Move,
Shift,
…

Name	Action
Op Dest, Src1, Src2	*RF (Dest) <- RF[Src1] Op RF[Src2]*
Op Dest, Src1, Src2	*RF (Dest) <- RF[Src1] Op Constant*
Move Dest, Src1	*RF (Dest) <- RF[Src1]*
Shift Dest, Src1, Constant	*RF (Dest) <- RF[Src1] shift Constant*

(b) Memory instructions:

Load,
Store,
…

Name	Action
L immU Dest	*RF [Dest(31…16)] <- Offset*
L immL Dest	*RF [Dest(15…0)] <- Offset*
L rel Dest, Src2, Offset	*RF [Dest] <- Mem[RF[Src2] + Offset]*
S rel Src1, Src2, Offset	*Mem[RF[Src2] + Offset] <- RF [Dest]*

(c) Control instructions:

Jump,
Branch,
…

Name	Action
Jump Offset	*PC <- PC + offset*
Jump Src2, Offset	*PC <- RF[Src2] + offset*
Brel Src1, Src2, Offset	*PC <- PC+1 if RF[Src1] rel RF[Src2]* *PC <- PC+Offset if RF[Src1] not rel RF[Src2]*

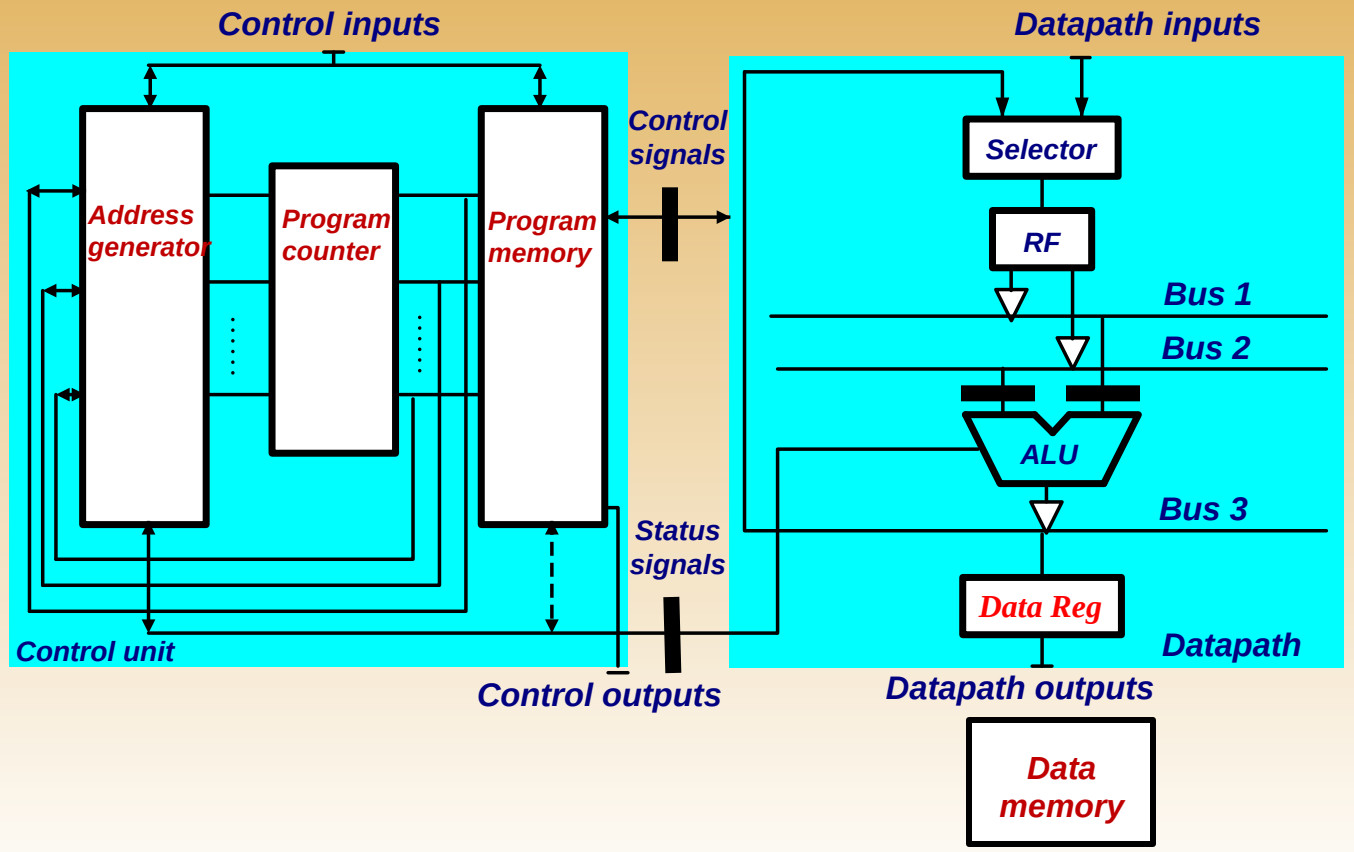
(d) Miscellaneous instructions:

No-op,
Clear,
Set,
Reset,
…

Name	Action
No-op	*Do nothing*
Clear Dest	*RF [Dest] <- 0*
Sstat Dest	*status [Dest] <- 1*
Rstat Dest	*status [Dest] <- 0*

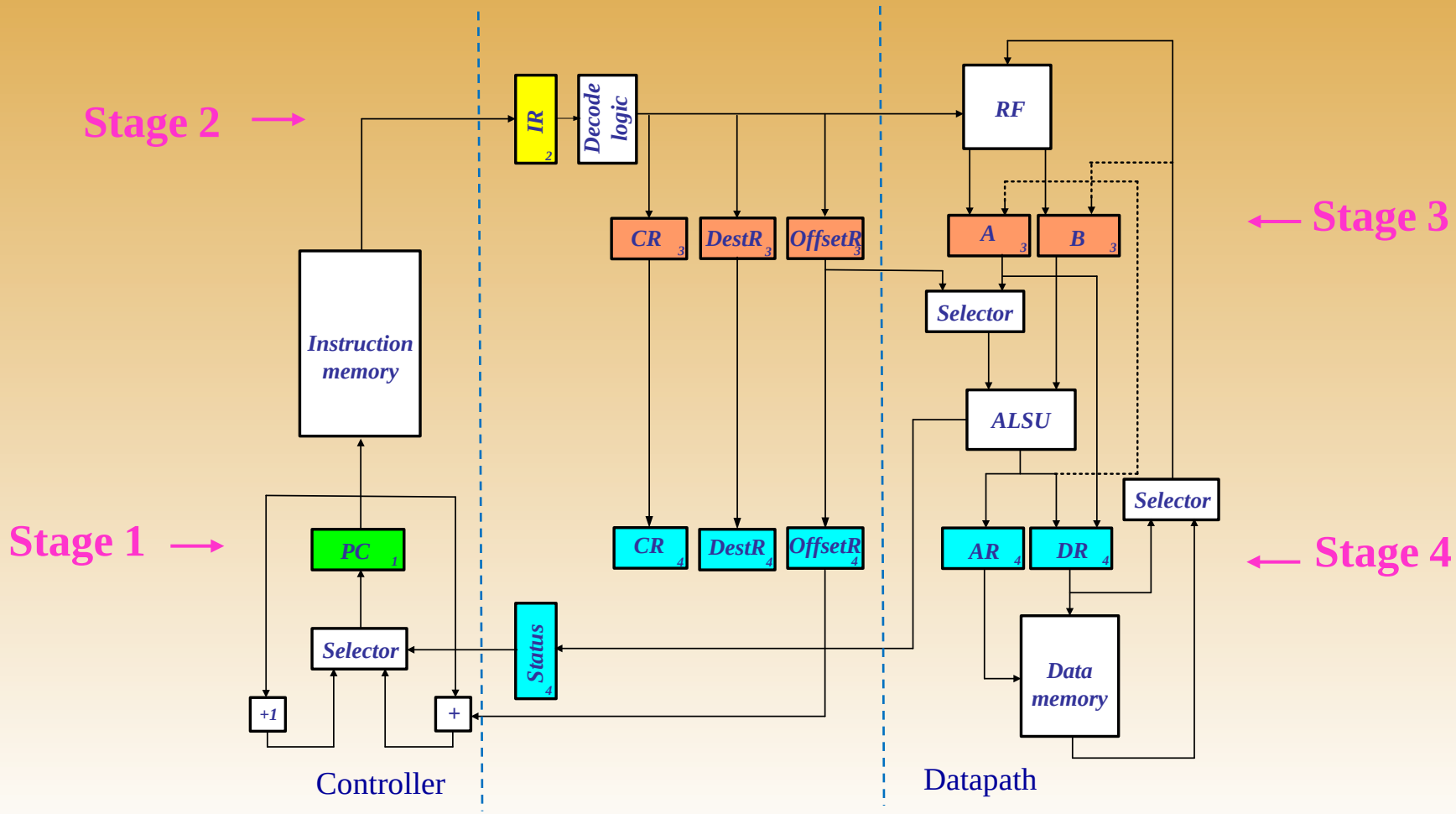
Old Controller/Datapath diagram

- Four stage control pipeline
- Two stage datapath pipeline



Processor block diagram

- Four stage pipeline
- Separate instruction and data memories
- Control register in each pipeline stage
- Pipeline stalling for control instruction



Program execution for a basic block

$x = a + b$

$y = b - c$

$z = c + d$

Source program

Address	Instruction
100	Load a, base, off
101	Load b, base, off
102	Load c, base, off
103	Load d, base, off
104	Add x, a, b
105	Sub y, b, c
106	Add z, c, d
107	Store x, base, off
108	Store y, base, off
109	Store z, base, off
110	—

Assembly program

Clock cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Read PC	100	101	102	103	104	105	106	107	108	109	110			
Write IR	Load	Load	Load	Load	Add	Sub	Add	Store	Store	Store				
Write A						a	b	c	x	y	z			
Write B		base	base	base	base	b	c	d	base	base	base			
Write AR			base +off	base +off	base +off	base +off				base +off		base +off		
Write DR							a+b	b-c	c+d	x	y	z		
Write RF				a	b	c	d	x	y	z				
Write Mem											x	y	z	
Write PC	101	102	103	104	105	106	107	108	109	110	111	112	113	114

Timing diagram

Program execution with data dependencies

sum = a + b

total = sum + c

Source program

Address	Instruction
100	Load a, base, offa
101	Load b, base, offb
102	Load c, base, offc
103	No-op
104	Add sum, a, b
105	No-op
106	No-op
107	Add total, c, sum
108	No-op
109	No-op
110	Store total, base, offt

Assembly program

Clock cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Read PC	100	101	102	103	104	105	106	107	108	109	110			
Write IR	Load	Load	Load	No-op	Add	No-op	No-op	Add	No-op	No-op	Store			
Write A						a			sum			total		
Write B		base	base	base		b			c			base		
Write AR			base	base	base								base	
			+offa	+offb	+offc								+offt	
Write DR							a+b			sum+c			total	
Write RF				a	b	c		sum			total			
Write Mem														total
Write PC	101	102	103	104	105	106	107	108	109	110	111	112	113	114

Timing diagram

Program execution with data-forwarding

$$sum = a + b$$

$$total = sum + c$$

Source program

Address	Instruction
100	Load a, base, off
101	Load b, base, off
102	Load c, base, off
103	Add sum, a, b
104	Add total, c, sum
105	Store total, base, off

Assembly program

Clock cycle	0	1	2	3	4	5	6	7	8
Read PC	100	101	102	103	104	105			
Write IR	Load	Load	Load	Add	Add	Store			
Write A					a	sum	sum+c		
Write B		base	base	base	b	c	Base		
Write AR			base +off	base +off	base +off			base +off	
Write DR						sum	sum+c	sum+c	
Write status									
Write RF				a	b	c	sum	sum+c	
Write Mem									total
Write PC	101	102	103	104	105	106	107	108	109

Timing diagram

Program execution without branch prediction

```
If a≥b then
    begin
        max = a
        min = b
    end
else
    begin
        max = b
        min = a
    end
endif
```

Source program

Address	Instruction
100	Bgoeq a, b, +10
101	No-op
102	No-op
103	No-op
104	Move max, b
105	Move min, a
106	Jump + 6
107	No-op
108	No-op
109	No-op
110	Move max, a
111	Move min, b
112	—

Assembly program

Clock cycle	0	1	2	3	4	5	6	7	8	9	10
Read PC	100	101	102	103	104	105	106	107	108	109	112
Write IR	Bgoeq	No-op	No-op	No-op	Move	Move	Jump	No-op	No-op	No-op	
Write A		a				b	a				
Write B		b									
Write AR											
Write DR							b	a			
Write status			a≥b								
Write RF								max	min		
Write Mem											
Write PC	101	102	103	104	105	106	107	108	109	112	

Timing diagram when "else" branch is taken

Clock cycle	0	1	2	3	4	5	6	7	8	9	10
Read PC	100	101	102	103	110	111	112				
Write IR	Bgoeq	No-op	No-op	No-op	Move	Move					
Write A		a				a	b				
Write B		b									
Write AR											
Write DR							a	b			
Write status			a≥b								
Write RF								max	min		
Write Mem											
Write PC	101	102	103	110	111	112					

Timing diagram when "then" branch is taken

Program execution with branch prediction

```
If a≥b then
    begin
        max = a
        min = b
    end
else
    begin
        max = b
        min = a
    end
endif
```

Source program

Address	Instruction
100	Bgoeq a, b, +4
101	Move max, b
102	Move min, a
103	Jump + 3
104	Move max, a
105	Move min, b
106	—

Assembly program

Clock cycle	0	1	2	3	4	5	6	7	8
Read PC	100	101	102	103	104	105	106	106	
Write IR	Bgoeq	Move	Move	Jump	Move	Move			
Write A		a	b	a		b			
Write B		b							
Write AR									
Write DR				b	a				
Write status			a≥b						
Write RF					max	min			
Write Mem									
Write PC	101	102	103	104	105	106	106		

Pipe Flush

Timing diagram when "else" branch is taken

Clock cycle	0	1	2	3	4	5	6	7	8
Read PC	100	101	102	103	104	105	106	106	
Write IR	Bgoeq	Move	Move	Jump	Move	Move			
Write A		a	b	a		a	b		
Write B		b							
Write AR									
Write DR				b			a	b	
Write status			a≥b						
Write RF								max	min
Write Mem									
Write PC	101	102	103	104	105	106			

Pipe Flush

Timing diagram when "then" branch is taken

Summary

- **We introduced new concepts:**
  - **Instruction sets**
  - **Instruction types**
  - **Addressing modes**
  - **Instruction cycle**
  - **Processor design flow**
- **We have demonstrated processor operation:**
  - **32-bit RISC design**
    - ***With*** **instruction pipelining, data-forwarding, branch prediction**