

RISC, CISC, and ISA Variations

Prof. Hakim Weatherspoon

CS 3410, Spring 2015

Computer Science

Cornell University

See P&H Appendix 2.16 – 2.18, and 2.21

Announcements

There *is* a Lab Section this week, C-Lab2

Project1 (PA1) is due next Monday, March 9th

Prelim today

Starts at **7:30pm** sharp

Go to location based on netid

[a-g]* → MRS146: Morrison Hall 146

[h-l]* → RRB125: Riley-Robb Hall 125

[m-n]* → RRB105: Riley-Robb Hall 105

[o-s]* → MVRG71: M Van Rensselaer Hall G71

[t-z]* → MVRG73: M Van Rensselaer Hall G73

Announcements

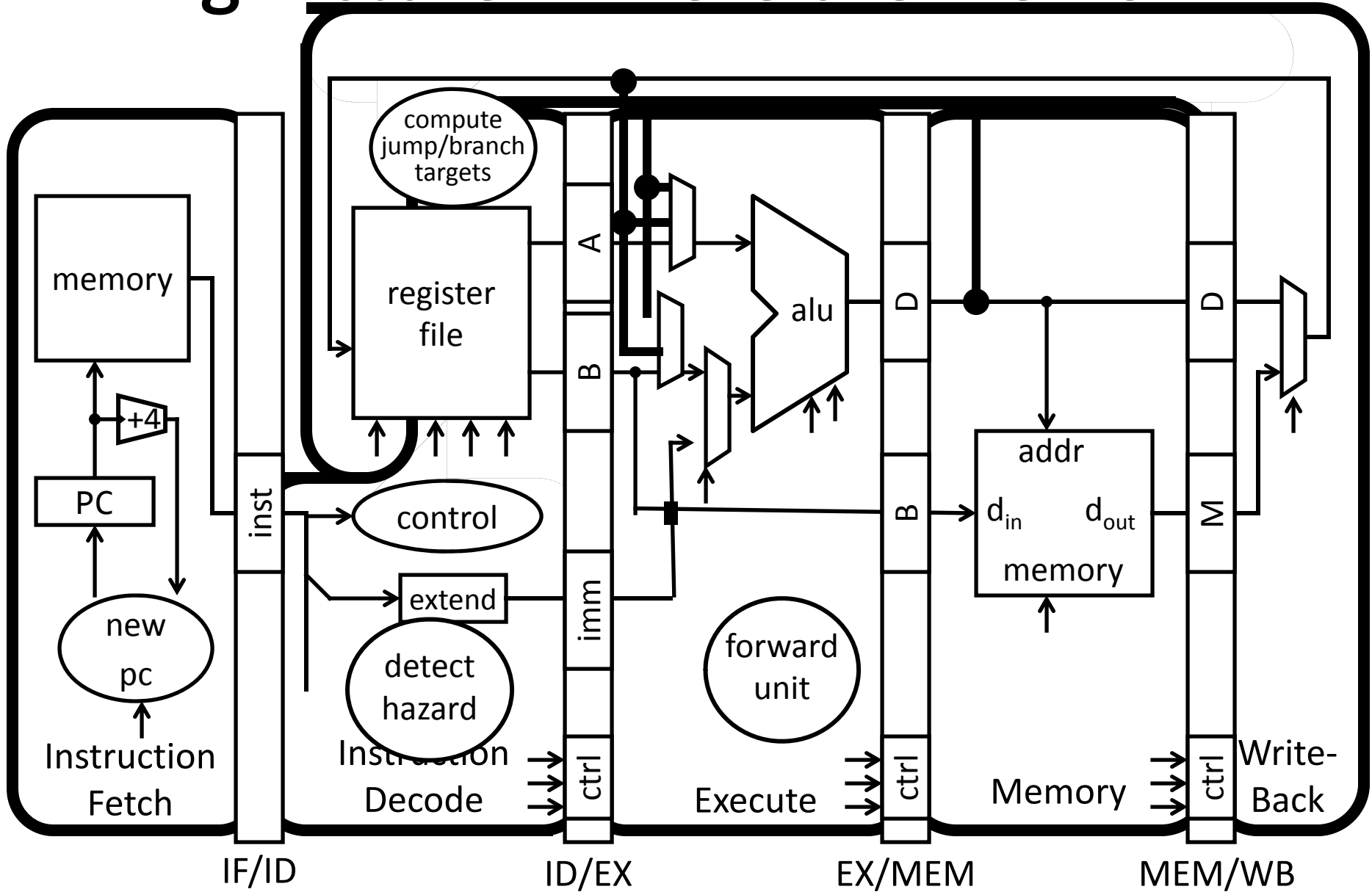
Prelim1 *today*:

- Time: We will start at 7:30pm sharp, so come early
- Location: on previous slide
- Closed Book
 - Cannot use electronic device or outside material
- Practice prelims are online in CMS

Material covered everything up to end of this week

- Everything up to and including data hazards
- Appendix B (logic, gates, FSMs, memory, ALUs)
- Chapter 4 (pipelined [and non] MIPS processor with hazards)
- Chapters 2 (Numbers / Arithmetic, simple MIPS instructions)
- Chapter 1 (Performance)
- HW1, Lab0, Lab1, Lab2, C-Lab0, C-Lab1

Big Picture: Where are we now?

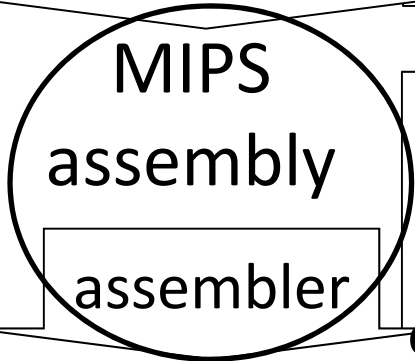


Big Picture: Where are we going?

C

```
int x = 10;
x = 2 * x + 15;
```

compiler



```
addi r5, r0, 10
mulr r5, r5, 2
addi r5, r5, 15
```

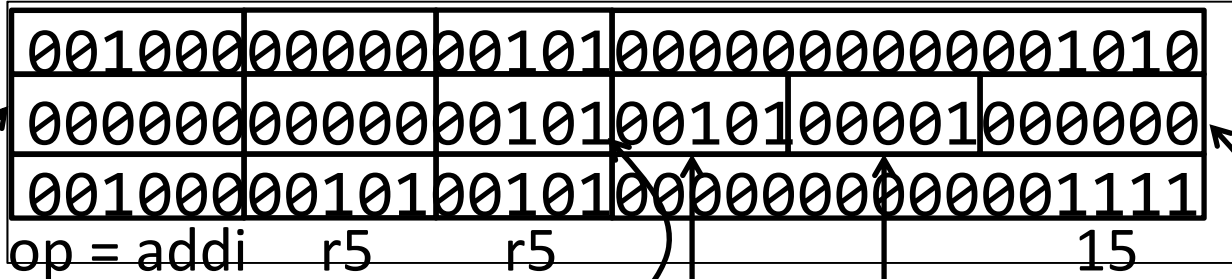
r0 = 0

r5 = r0 + 10

r5 = r5 << 1 # r5 = r5 * 2

r5 = r5 + 15

machine code



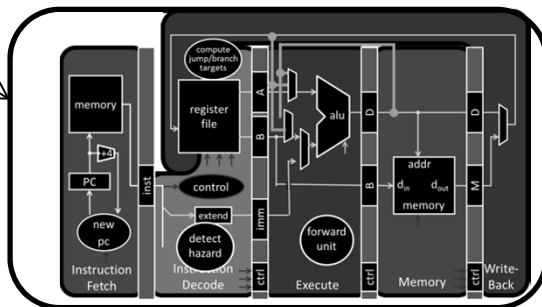
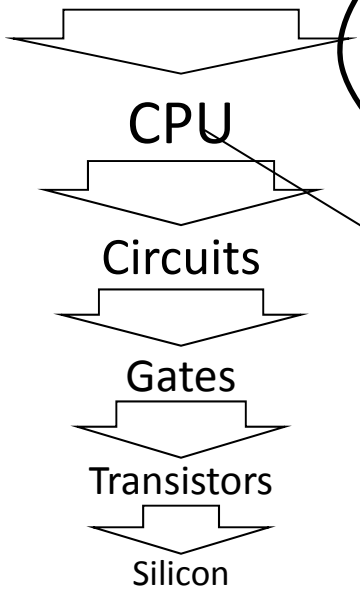
op = r-type

r5

r5

shamt=1

func=sll



Big Picture: Where are we going?

C

compiler

```
int x = 10;
x = 2 * x + 15;
```

High Level Languages

MIPS
assembly

```
addi r5, r0, 10
mulr r5, r5, 2
addi r5, r5, 15
```

assembler

machine
code

```
001000000000010100000000000001010
000000000000001010010100001000000
001000001010010100000000000001111
```

Instruction Set Architecture (ISA)

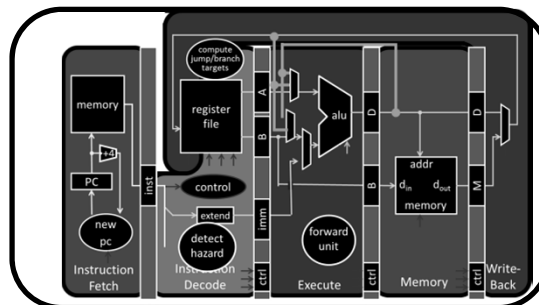
CPU

Circuits

Gates

Transistors

Silicon



Goals for Today

Instruction Set Architectures

- ISA Variations, and CISC vs RISC

Next Time

- Program Structure and Calling Conventions

Next Goal

Is MIPS the only possible instruction set architecture (ISA)?

What are the alternatives?

Instruction Set Architecture Variations

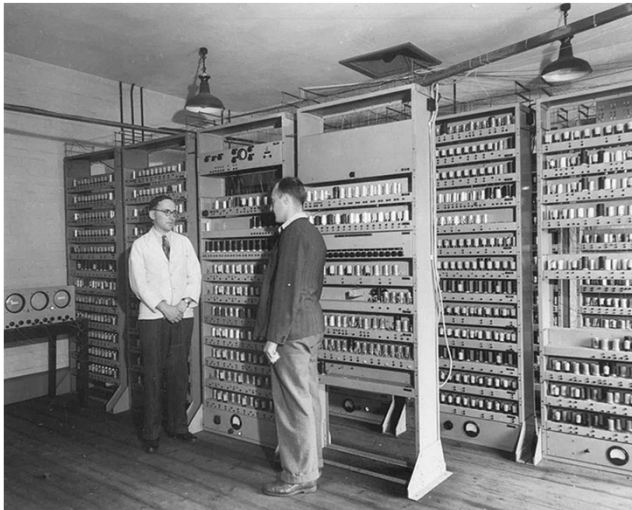
ISA defines the permissible instructions

- MIPS: load/store, arithmetic, control flow, ...
- ARMv7: similar to MIPS, but more shift, memory, & conditional ops
- ARMv8 (64-bit): even closer to MIPS, no conditional ops
- VAX: arithmetic on memory or registers, strings, polynomial evaluation, stacks/queues, ...
- Cray: vector operations, ...
- x86: a little of everything

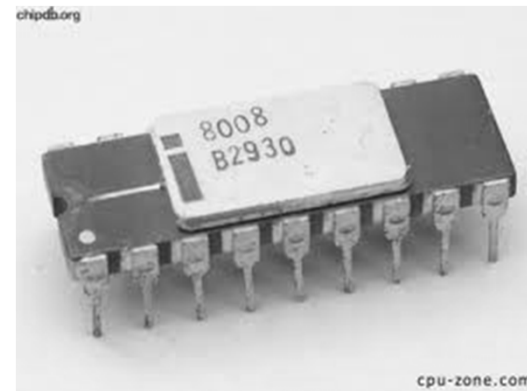
Brief Historical Perspective on ISAs

Accumulators

- Early stored-program computers had ***one*** register!



EDSAC (Electronic Delay Storage Automatic Calculator) in 1949



Intel 8008 in 1972
was an accumulator

- One register is two registers short of a MIPS instruction!
- Requires a memory-based operand-addressing mode
 - Example Instructions: `add 200`
 - Add the accumulator to the word in memory at address 200
 - Place the sum back in the accumulator

Brief Historical Perspective on ISAs

Next step, more registers...

- Dedicated registers
 - E.g. indices for array references in data transfer instructions, separate accumulators for multiply or divide instructions, top-of-stack pointer.



Intel 8086
“extended accumulator”
Processor for IBM PCs

- Extended Accumulator
 - One operand may be in memory (like previous accumulators).
 - Or, all the operands may be registers (like MIPS).

Brief Historical Perspective on ISAs

Next step, more registers...

- General-purpose registers
 - Registers can be used for any purpose
 - E.g. MIPS, ARM, x86
- *Register-memory* architectures
 - One operand may be in memory (e.g. accumulators)
 - E.g. x86 (i.e. 80386 processors)
- *Register-register* architectures (aka load-store)
 - All operands **must** be in registers
 - E.g. MIPS, ARM

Takeaway

The number of available registers greatly influenced the instruction set architecture (ISA)

Machine	Num General Purpose Registers	Architectural Style	Year
EDSAC	1	Accumulator	1949
IBM 701	1	Accumulator	1953
CDC 6600	8	Load-Store	1963
IBM 360	18	Register-Memory	1964
DEC PDP-8	1	Accumulator	1965
DEC PDP-11	8	Register-Memory	1970
Intel 8008	1	Accumulator	1972
Motorola 6800	2	Accumulator	1974
DEC VAX	16	Register-Memory, Memory-Memory	1977
Intel 8086	1	Extended Accumulator	1978
Motorola 6800	16	Register-Memory	1980
Intel 80386	8	Register-Memory	1985
ARM	16	Load-Store	1985
MIPS	32	Load-Store	1985
HP PA-RISC	32	Load-Store	1986
SPARC	32	Load-Store	1987
PowerPC	32	Load-Store	1992
DEC Alpha	32	Load-Store	1992
HP/Intel IA-64	128	Load-Store	2001
AMD64 (EMT64)	16	Register-Memory	2003

Takeaway

The number of available registers greatly influenced the instruction set architecture (ISA)

Machine	Number of general-purpose registers	Architectural style	Year
EDSAC	1	Accumulator	1949
IBM 701	1	Accumulator	1953
CDC 6600	8	Load-store	1963
IBM 360	16	Register-memory	1964
DEC PDP-8	1	Accumulator	1965
DEC PDP-11	8	Register-memory	1970
Intel 8008	1	Accumulator	1972
Motorola 6800	2	Accumulator	1974
DEC VAX	16	Register-memory, memory-memory	1977
Intel 8086	1	Extended accumulator	1978
Motorola 68000	16	Register-memory	1980
Intel 80386	8	Register-memory	1985
ARM	16	Load-store	1985
MIPS	32	Load-store	1985
HP PA-RISC	32	Load-store	1986
SPARC	32	Load-store	1987
PowerPC	32	Load-store	1992
DEC Alpha	32	Load-store	1992
HP/Intel IA-64	128	Load-store	2001
AMD64 (EMT64)	16	Register-memory	2003

Next Goal

How to compute with limited resources?

i.e. how do you design your ISA if you have limited resources?

People programmed in assembly and machine code!

- Needed as many addressing modes as possible
- Memory was (and still is) slow

CPUs had relatively few registers

- Register's were more “expensive” than external mem
- Large number of registers requires many bits to index

Memories were small

- Encouraged highly encoded microcodes as instructions
- Variable length instructions, load/store, conditions, etc

People programmed in assembly and machine code!

E.g. x86

- > 1000 instructions!
 - 1 to 15 bytes each
 - E.g. dozens of add instructions
- operands in dedicated registers, general purpose registers, memory, on stack, ...
 - can be 1, 2, 4, 8 bytes, signed or unsigned
- 10s of addressing modes
 - e.g. Mem[segment + reg + reg*scale + offset]

E.g. VAX

- Like x86, arithmetic on memory or registers, but also on strings, polynomial evaluation, stacks/queues, ...

Complex Instruction Set Computers (CISC)

Takeaway

The number of available registers greatly influenced the instruction set architecture (ISA)

Complex Instruction Set Computers were very complex

- Necessary to reduce the number of instructions required to fit a program into memory.
- However, also greatly increased the complexity of the ISA as well.

Next Goal

How do we reduce the complexity of the ISA while maintaining or increasing performance?

Reduced Instruction Set Computer (RISC)

John Cock

- IBM 801, 1980 (started in 1975)
- Name 801 came from the bldg that housed the project
- Idea: Possible to make a very small and very fast core
- Influences: Known as “the father of RISC Architecture”. Turing Award Recipient and National Medal of Science.



Reduced Instruction Set Computer (RISC)

Dave Patterson

- RISC Project, 1982
- UC Berkeley
- RISC-I: ½ transistors & 3x faster
- Influences: Sun SPARC, namesake of industry



John L. Hennessy

- MIPS, 1981
- Stanford
- Simple pipelining, keep full
- Influences: MIPS computer system, PlayStation, Nintendo



Reduced Instruction Set Computer (RISC)

Dave Patterson

- RISC Project, 1982
- UC Berkeley
- RISC-I: ½ transistors & 3x faster
- Influences: Sun SPARC, namesake of industry



John L. Hennessy

- MIPS, 1981
- Stanford
- Simple pipelining, keep full
- Influences: MIPS computer system, PlayStation, Nintendo



Reduced Instruction Set Computer (RISC)

MIPS Design Principles

Simplicity favors regularity

- 32 bit instructions

Smaller is faster

- Small register file

Make the common case fast

- Include support for constants

Good design demands good compromises

- Support for different type of interpretations/classes

Reduced Instruction Set Computer

MIPS = Reduced Instruction Set Computer (RISC)

- \approx 200 instructions, 32 bits each, 3 formats
- all operands in registers
 - almost all are 32 bits each
- \approx ① addressing mode: Mem[reg + imm]

x86 = Complex Instruction Set Computer (CISC)

- $>$ 1000 instructions, 1 to 15 bytes each
- operands in dedicated registers, general purpose registers, memory, on stack, ...
 - can be 1, 2, 4, 8 bytes, signed or unsigned
- 10s of addressing modes
 - e.g. Mem[segment + reg + reg*scale + offset]

RISC vs CISC

RISC Philosophy

Regularity & simplicity

Leaner means faster

Optimize the
common case

Energy efficiency

Embedded Systems

Phones/Tablets

CISC Rebuttal

Compilers can be smart

Transistors are plentiful

Legacy is important

Code size counts

Micro-code!

Desktops/Servers

ARMDroid vs WinTel

- Android OS on ARM processor



- Windows OS on Intel (x86) processor



Takeaway

The number of available registers greatly influenced the instruction set architecture (ISA)

Complex Instruction Set Computers were very complex

- Necessary to reduce the number of instructions required to fit a program into memory.
- However, also greatly increased the complexity of the ISA as well.

Back in the day... CISC was necessary because everybody programmed in assembly and machine code! Today, CISC ISA's are still dominant due to the prevalence of x86 ISA processors. However, RISC ISA's today such as ARM have an ever increasing market share (of our everyday life!).

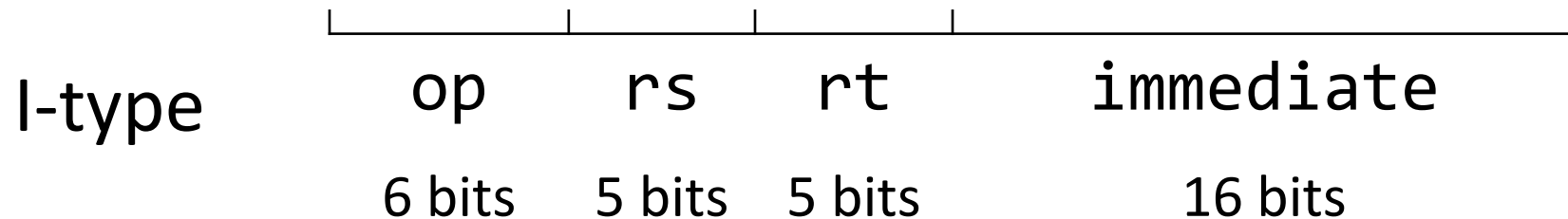
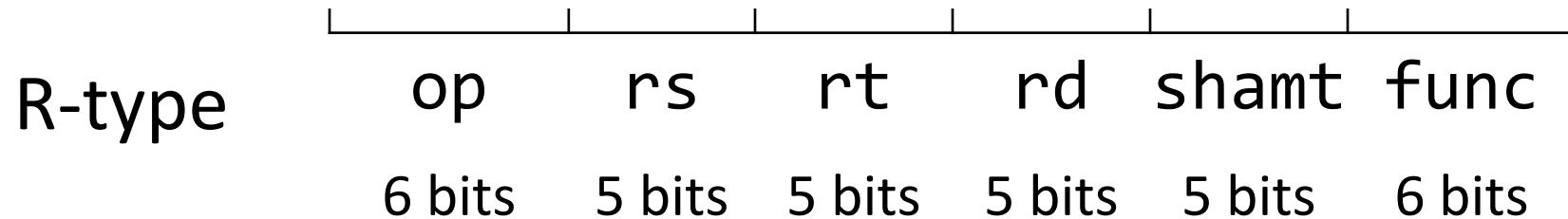
ARM borrows a bit from both RISC and CISC.

Next Goal

How does MIPS and ARM compare to each other?

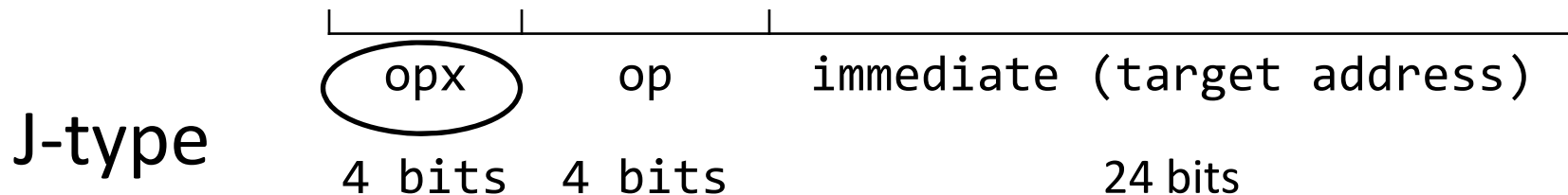
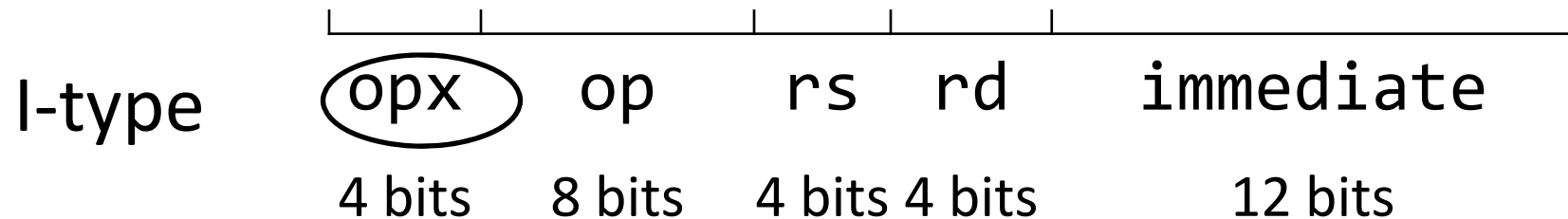
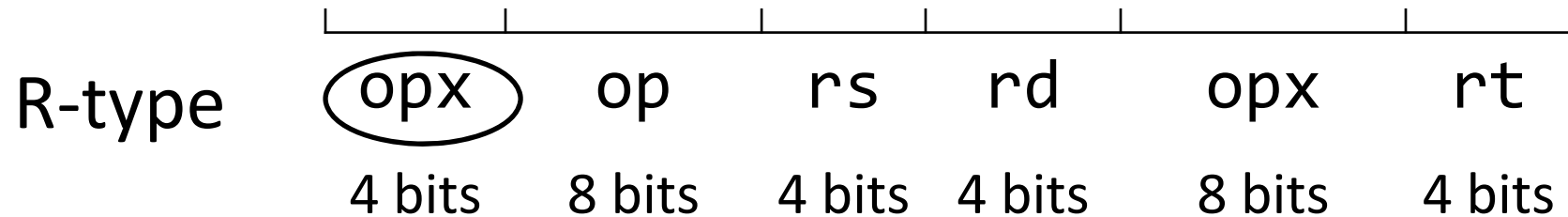
MIPS instruction formats

All MIPS instructions are 32 bits long, has 3 formats



ARMv7 instruction formats

All ARMv7 instructions are 32 bits long, has 3 formats



ARMv7 Conditional Instructions

- while(i != j) {
- if (i > j)
- i -= j;
- else
- j -= i;
- }

In MIPS, performance will be slow if code has a lot of branches

```
Loop: BEQ Ri, Rj, End       // if "NE" (not equal), then stay in loop
      SLT Rd, Rj, Ri       // "GT" if (i > j),
      BNE Rd, R0, Else     // ...
      SUB Ri, Ri, Rj       // if "GT" (greater than), i = i-j;
      J Loop
Else: SUB Rj, Rj, Ri       // or "LT" if (i < j)
      J Loop               // if "LT" (less than), j = j-i;
End:
```


ARMv7 Conditional Instructions

- while(i != j) {
- if (i > j)
- i -= j; In ARM, can avoid delay due to
- else Branches with conditional
- j -= i; instructions
- }

LOOP: CMP Ri, Rj

0	1	0	0
=	≠	<	>

 // set condition "NE" if (i != j)

// "GT" if (i > j),

// or "LT" if (i < j)

<table border="1" style="display: inline-table;"><tr><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td></tr><tr><td style="text-align: center;">=</td><td style="text-align: center;">≠</td><td style="text-align: center;"><</td><td style="text-align: center;">></td></tr></table>	0	0	0	1	=	≠	<	>	SUBGT Ri, Ri, Rj	// if "GT" (greater than), i = i-j;
0	0	0	1							
=	≠	<	>							
<table border="1" style="display: inline-table;"><tr><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">=</td><td style="text-align: center;">≠</td><td style="text-align: center;"><</td><td style="text-align: center;">></td></tr></table>	1	0	1	0	=	≠	<	>	SUBLE Rj, Rj, Ri	// if "LE" (less than or equal), j = j-i;
1	0	1	0							
=	≠	<	>							
<table border="1" style="display: inline-table;"><tr><td style="text-align: center;">0</td><td style="text-align: center;">1</td><td style="text-align: center;">0</td><td style="text-align: center;">0</td></tr><tr><td style="text-align: center;">=</td><td style="text-align: center;">≠</td><td style="text-align: center;"><</td><td style="text-align: center;">></td></tr></table>	0	1	0	0	=	≠	<	>	BNE loop	// if "NE" (not equal), then loop
0	1	0	0							
=	≠	<	>							

ARMv7: Other Cool operations

Shift one register (e.g. R_c) any amount

Add to another register (e.g. R_b)

Store result in a different register (e.g. R_a)

ADD R_a, R_b, R_c LSL #4

$R_a = R_b + R_c \ll 4$

$R_a = R_b + R_c \times 16$

ARMv7 Instruction Set Architecture

All ARMv7 instructions are 32 bits long, has 3 formats

Reduced Instruction Set Computer (RISC) properties

- Only Load/Store instructions access memory
- Instructions operate on operands in processor registers
- 16 registers

Complex Instruction Set Computer (CISC) properties

- Autoincrement, autodecrement, PC-relative addressing
- Conditional execution
- Multiple words can be accessed from memory with a single instruction (SIMD: single instr multiple data)

ARMv8 (64-bit) Instruction Set Architecture

All ARMv8 instructions are 64 bits long, has 3 formats

Reduced Instruction Set Computer (RISC) properties

- Only Load/Store instructions access memory
- Instructions operate on operands in processor registers
- **32** registers and r0 is always 0

NO MORE Complex Instruction Set Computer (CISC) properties

- NO Conditional execution
- NO Multiple words can be accessed from memory with a single instruction (SIMD: single instr multiple data)

Instruction Set Architecture Variations

ISA defines the permissible instructions

- MIPS: load/store, arithmetic, control flow, ...
- ARMv7: similar to MIPS, but more shift, memory, & conditional ops
- ARMv8 (64-bit): even closer to MIPS, no conditional ops
- VAX: arithmetic on memory or registers, strings, polynomial evaluation, stacks/queues, ...
- Cray: vector operations, ...
- x86: a little of everything

Next time

How do we coordinate use of registers?

Calling Conventions!

PA1 due next Tuesday

Prelim 1 Review Questions

Prelim 1

Prelim today

Starts at **7:30pm** sharp

Go to location based on netid

[a-g]* → MRS146: Morrison Hall 146

[h-l]* → RRB125: Riley-Robb Hall 125

[m-n]* → RRB105: Riley-Robb Hall 105

[o-s]* → MVRG71: M Van Rensselaer Hall G71

[t-z]* → MVRG73: M Van Rensselaer Hall G73

Prelim 1

Time: We will start at **7:30pm sharp**, so come early

Location: See previous slide

Closed Book

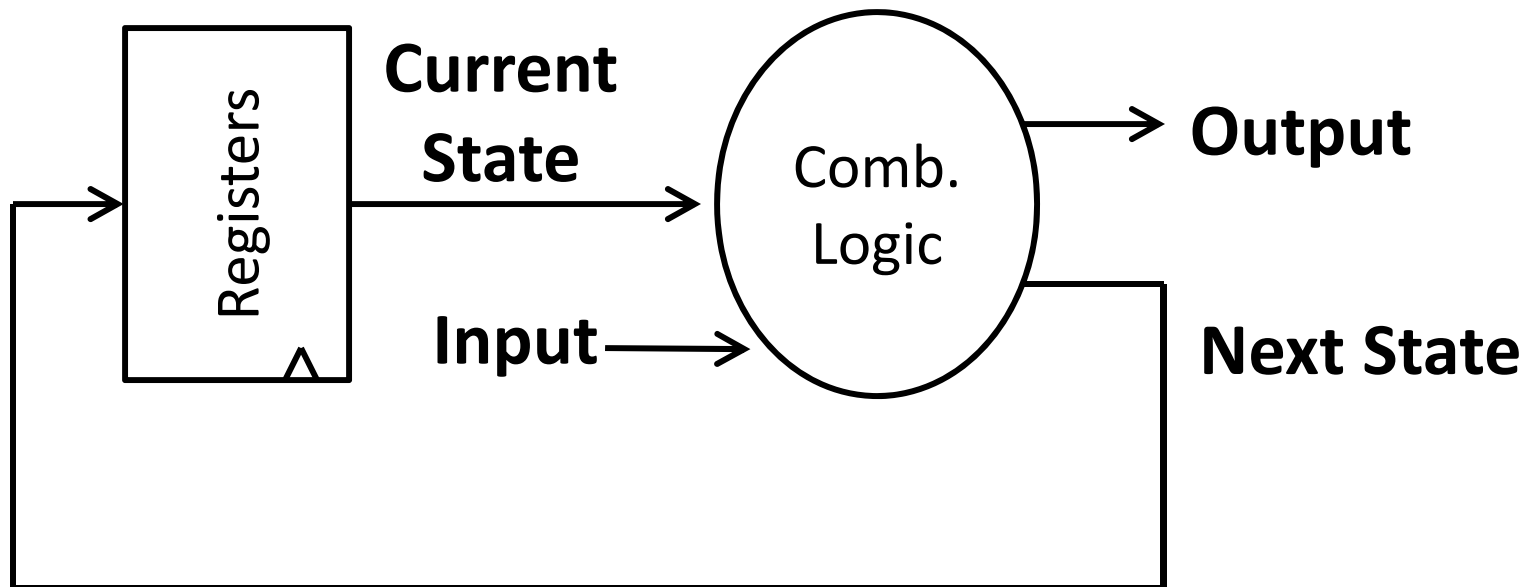
- Cannot use electronic device or outside material

Material covered everything up to end of last week

- Everything up to and including data hazards
- Appendix B (logic, gates, FSMs, memory, ALUs)
- Chapter 4 (pipelined [and non] MIPS processor with hazards)
- Chapters 2 (Numbers / Arithmetic, simple MIPS instructions)
- Chapter 1 (Performance)
- HW1, Lab0, Lab1, Lab2

Mealy Machine

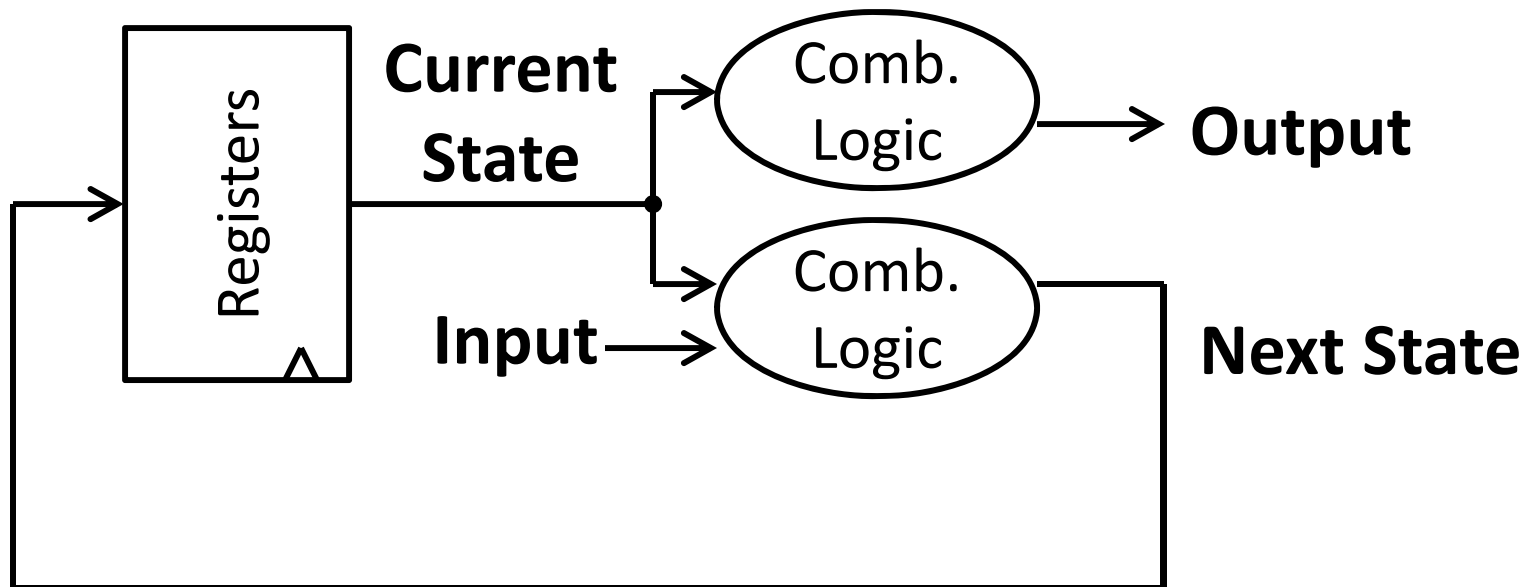
General Case: Mealy Machine



Outputs and next state depend on both current state and input

Moore Machine

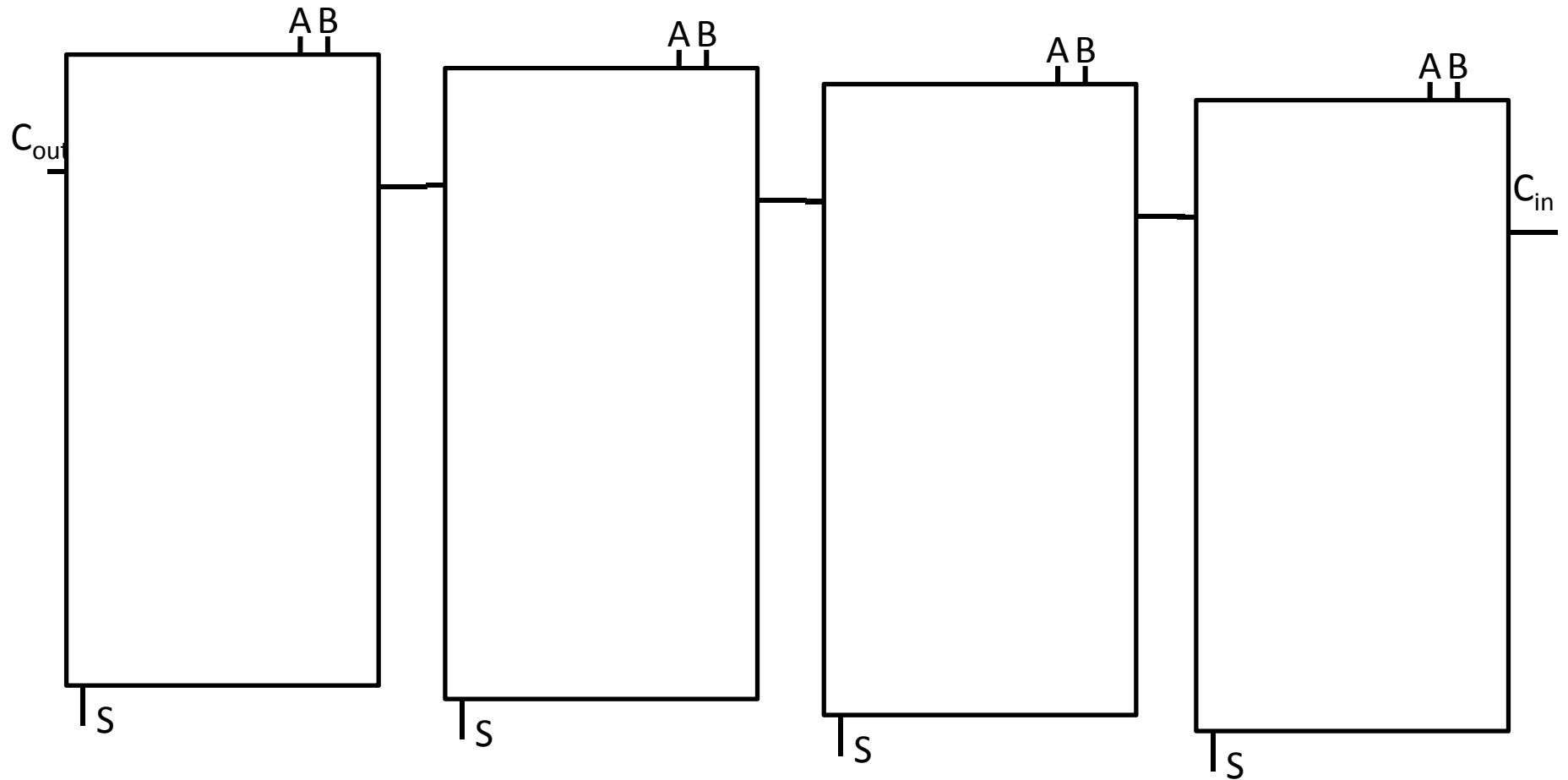
Special Case: Moore Machine



Outputs depend only on current state

Critical Path

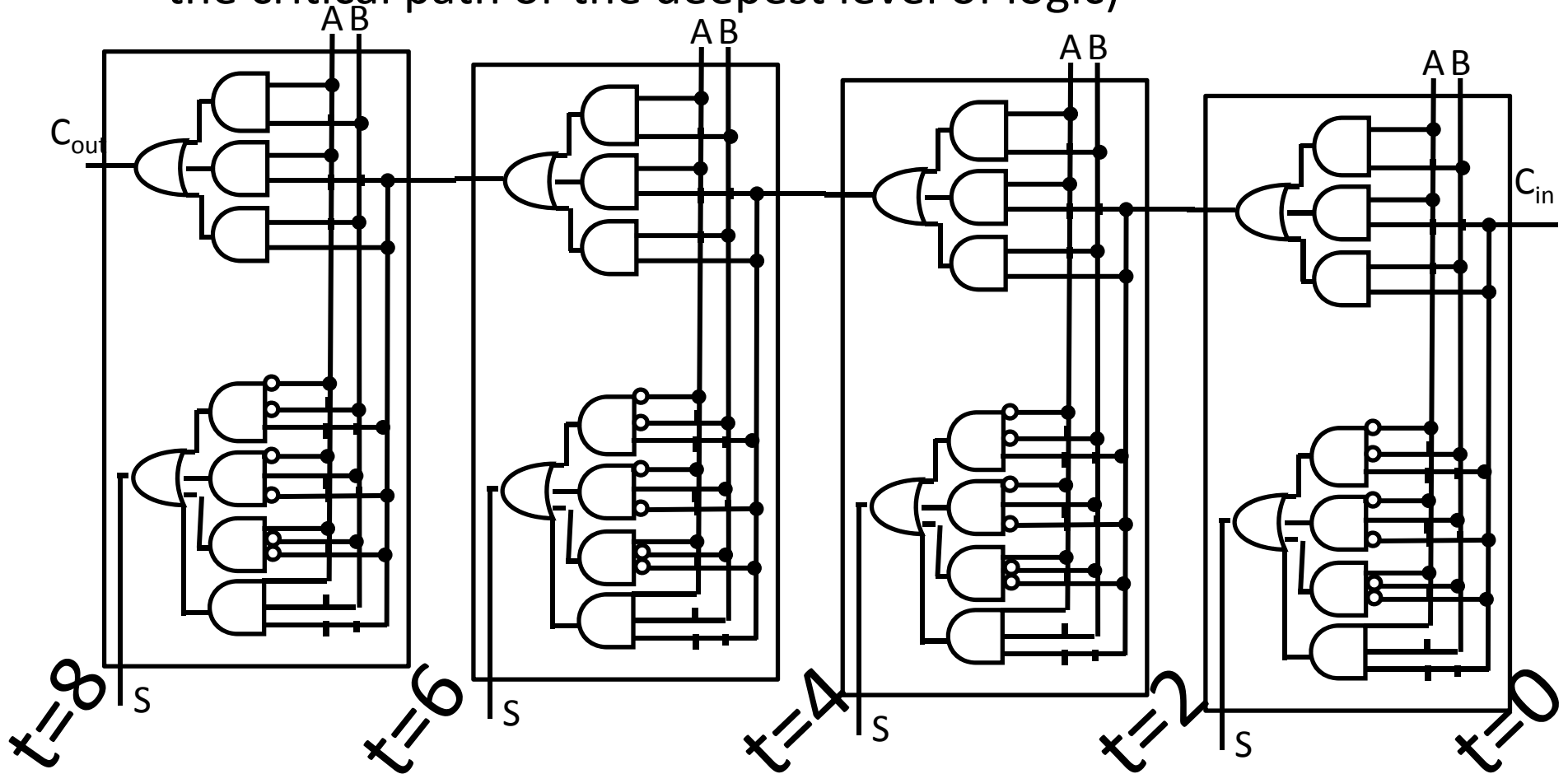
How long does it take to compute a result?



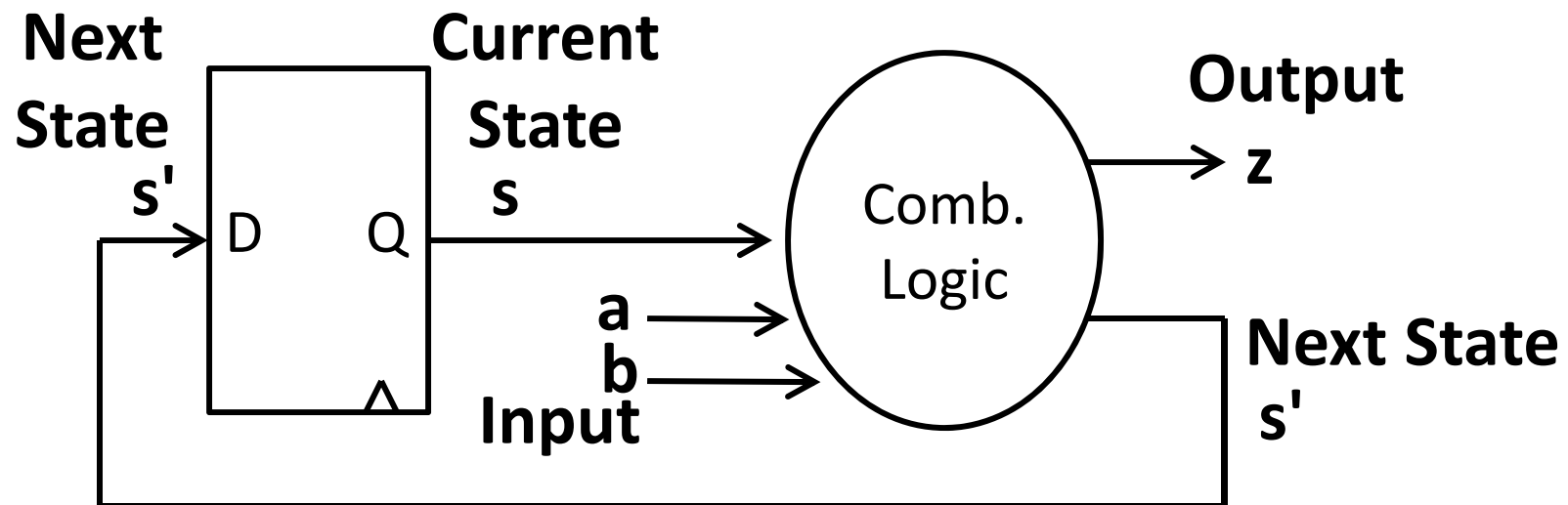
Critical Path

How long does it take to compute a result?

- Speed of a circuit is affected by the number of gates in series (on the critical path or the deepest level of logic)



Example: Mealy Machine



$$z = \bar{a}b\bar{s} + a\bar{b}\bar{s} + \bar{a}bs + abs$$

$$s' = ab\bar{s} + \bar{a}bs + a\bar{b}s + abs$$

Strategy:

- (1) Draw a state diagram (e.g. Mealy Machine)
- (2) Write output and next-state tables
- (3) Encode states, inputs, and outputs as bits
- (4) Determine logic equations for next state and outputs

Endianness

Endianness: Ordering of bytes within a memory word

Little Endian = least significant part first (MIPS, x86)

	1000	1001	1002	1003
as 4 bytes	0x78	0x56	0x34	0x12
as 2 halfwords	0x5678		0x1234	
as 1 word	0x12345678			

Big Endian = most significant part first (MIPS, networks)

	1000	1001	1002	1003
as 4 bytes	0x12	0x34	0x56	0x78
as 2 halfwords	0x1234		0x5678	
as 1 word	0x12345678			

Memory Layout

Examples (big/little endian):

r5 contains 5 (0x00000005)

SB r5, 2(r0)

LB r6, 2(r0)

R[r6] = 0x05

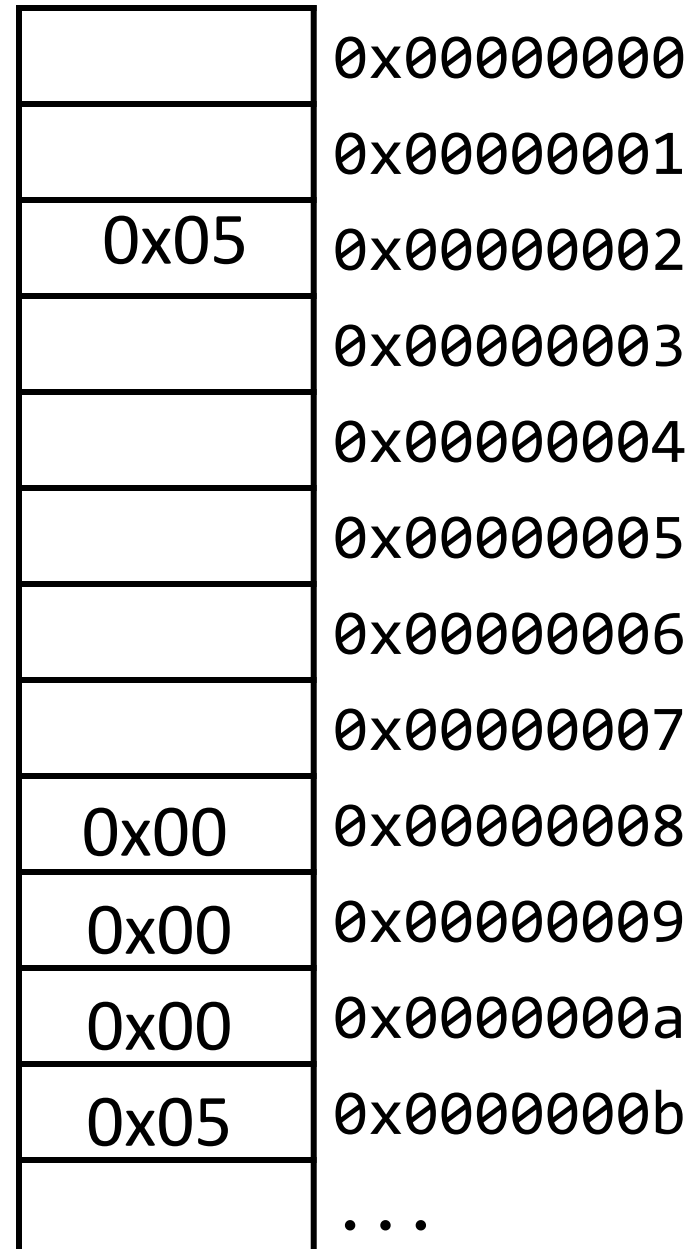
SW r5, 8(r0)

LB r7, 8(r0)

LB r8, 11(r0)

R[r7] = 0x00

R[r8] = 0x05



Memory Layout

Examples (big/little endian):

r5 contains 5 (0x00000005)

SB r5, 2(r0)

LB r6, 2(r0)

R[r6] = 0x00000005

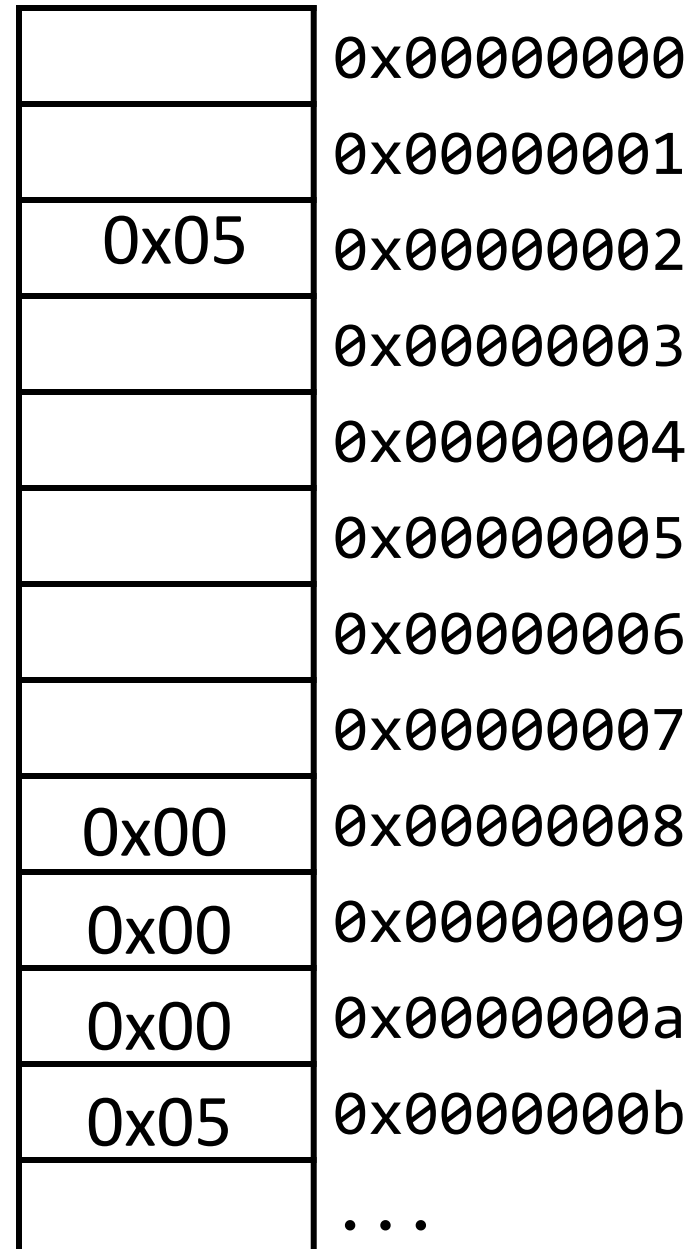
SW r5, 8(r0)

LB r7, 8(r0)

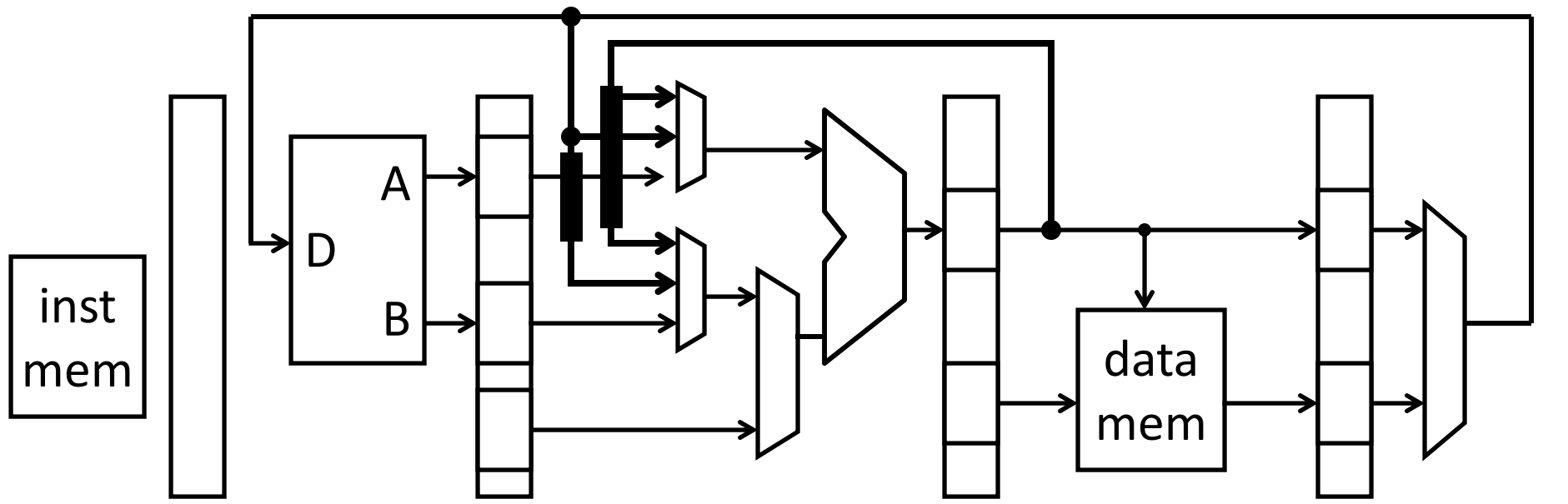
LB r8, 11(r0)

R[r7] = 0x00000000

R[r8] = 0x00000005



Forwarding Datapath 2

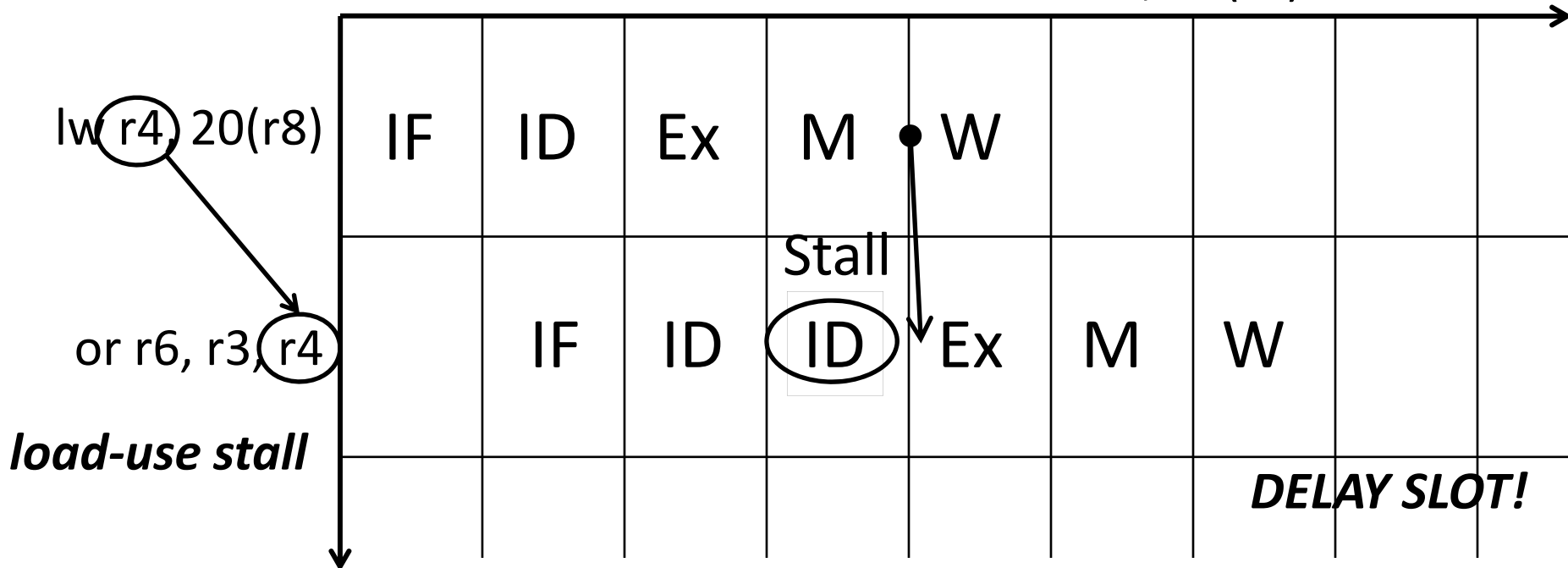
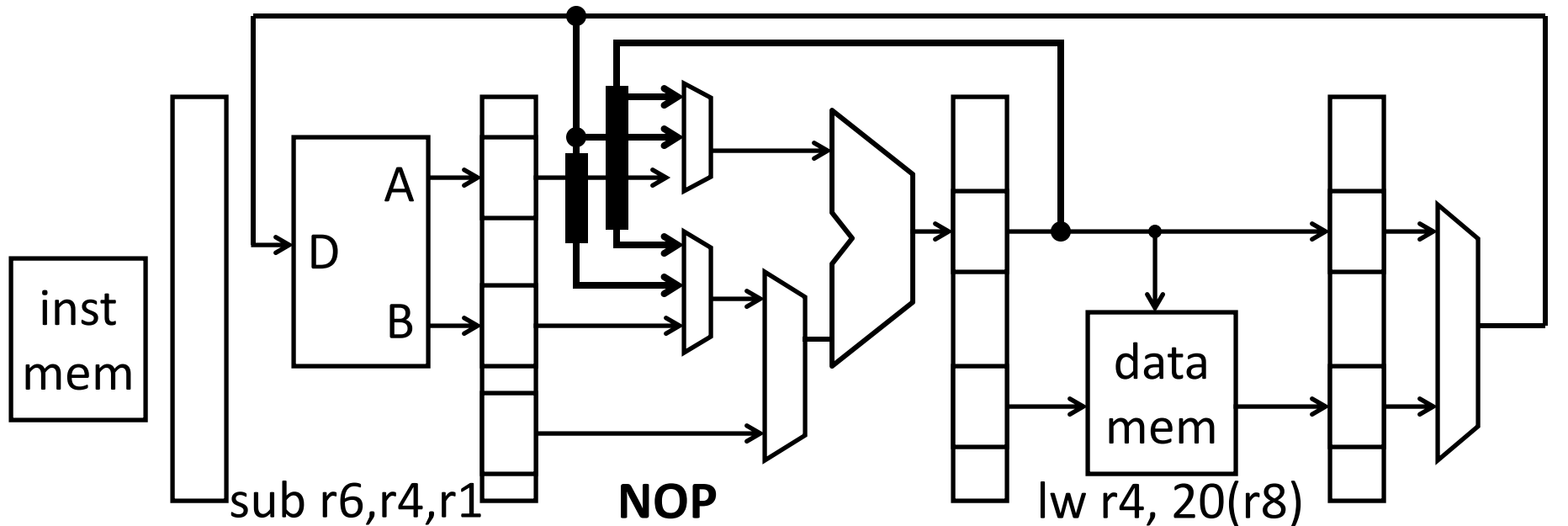


add r3, r1, r2	IF	ID	Ex	M	W			
sub r5, r3, r1		IF	ID	Ex	M	W		
or r6, r3, r4			IF	ID	Ex	M	W	

Arrows in the table indicate forwarding:

- From the **W** stage of the first instruction to the **Ex** stage of the second instruction.
- From the **W** stage of the second instruction to the **Ex** stage of the third instruction.

Memory Load Data Hazard



Quiz

add r3, r1, r2

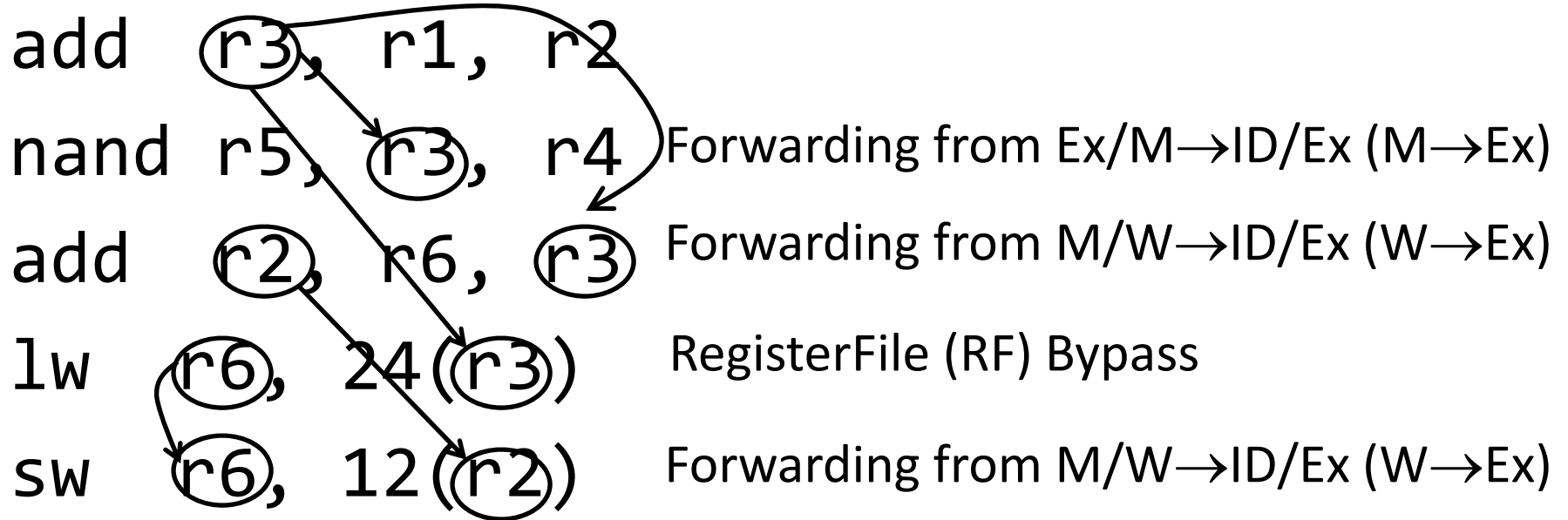
nand r5, r3, r4

add r2, r6, r3

lw r6, 24(r3)

sw r6, 12(r2)

Quiz



Stall

+ Forwarding from M/W→ID/Ex (W→Ex)

5 Hazards

Questions?