ENGN2219/COMP6719
Computer Systems & Organization

Convener: Shoaib Akram

shoaib.akram@anu.edu.au

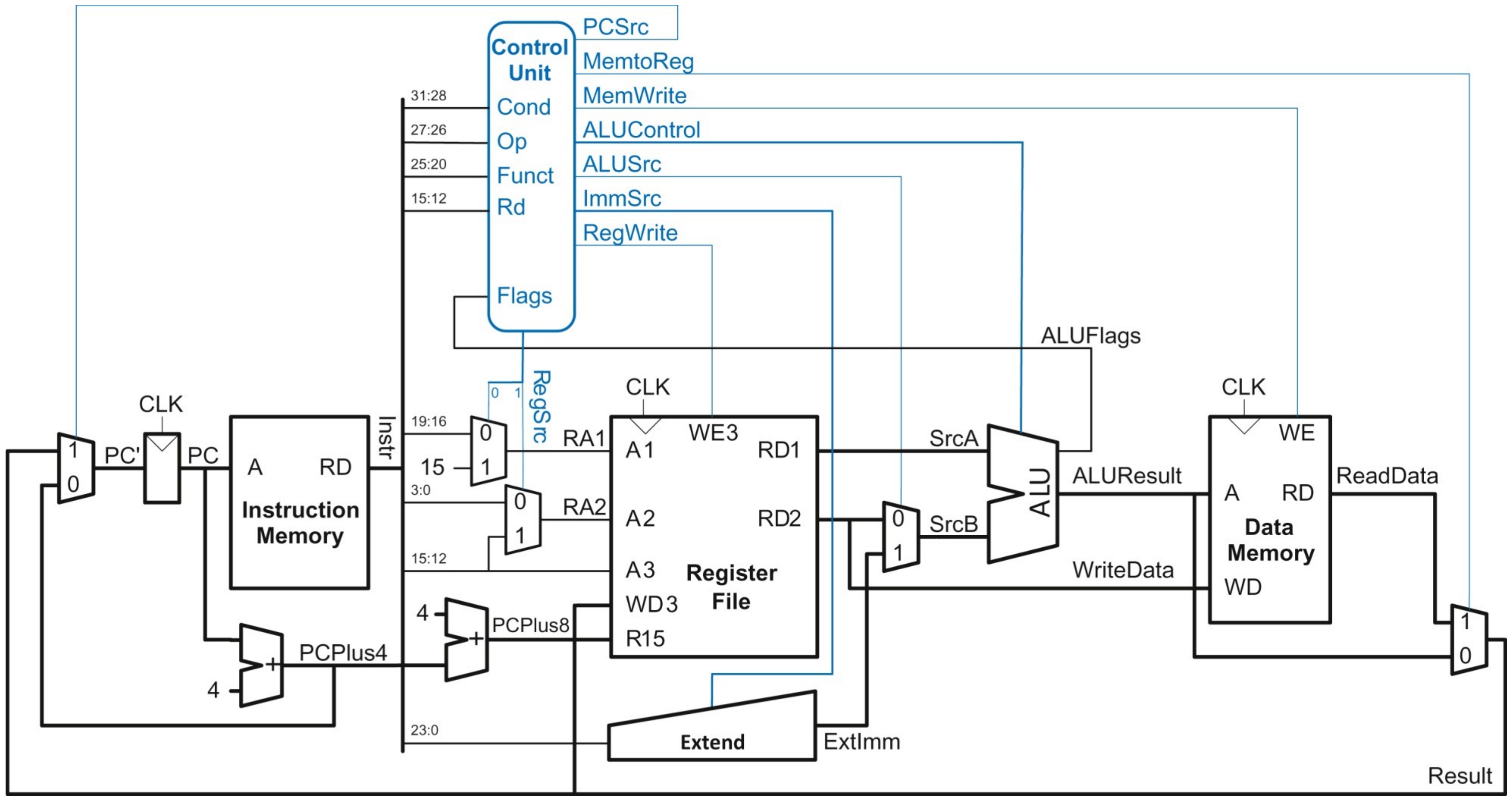
Australian National University

Plan: Week 6

Last week: Instruction Set Architecture (specification)

This Week: Microarchitecture (implementation)

Processor Operation: ORR

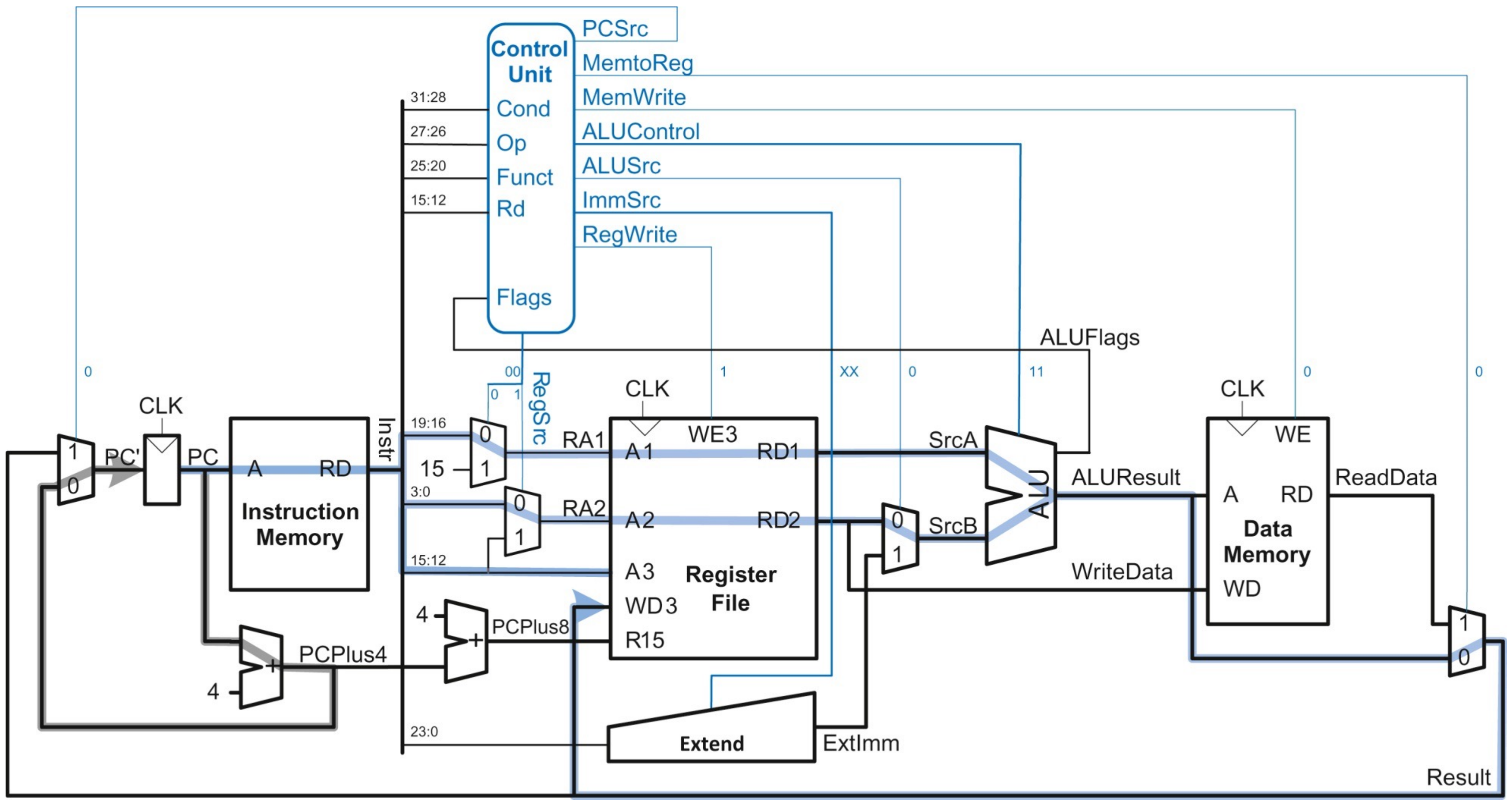


Processor Operation: ORR

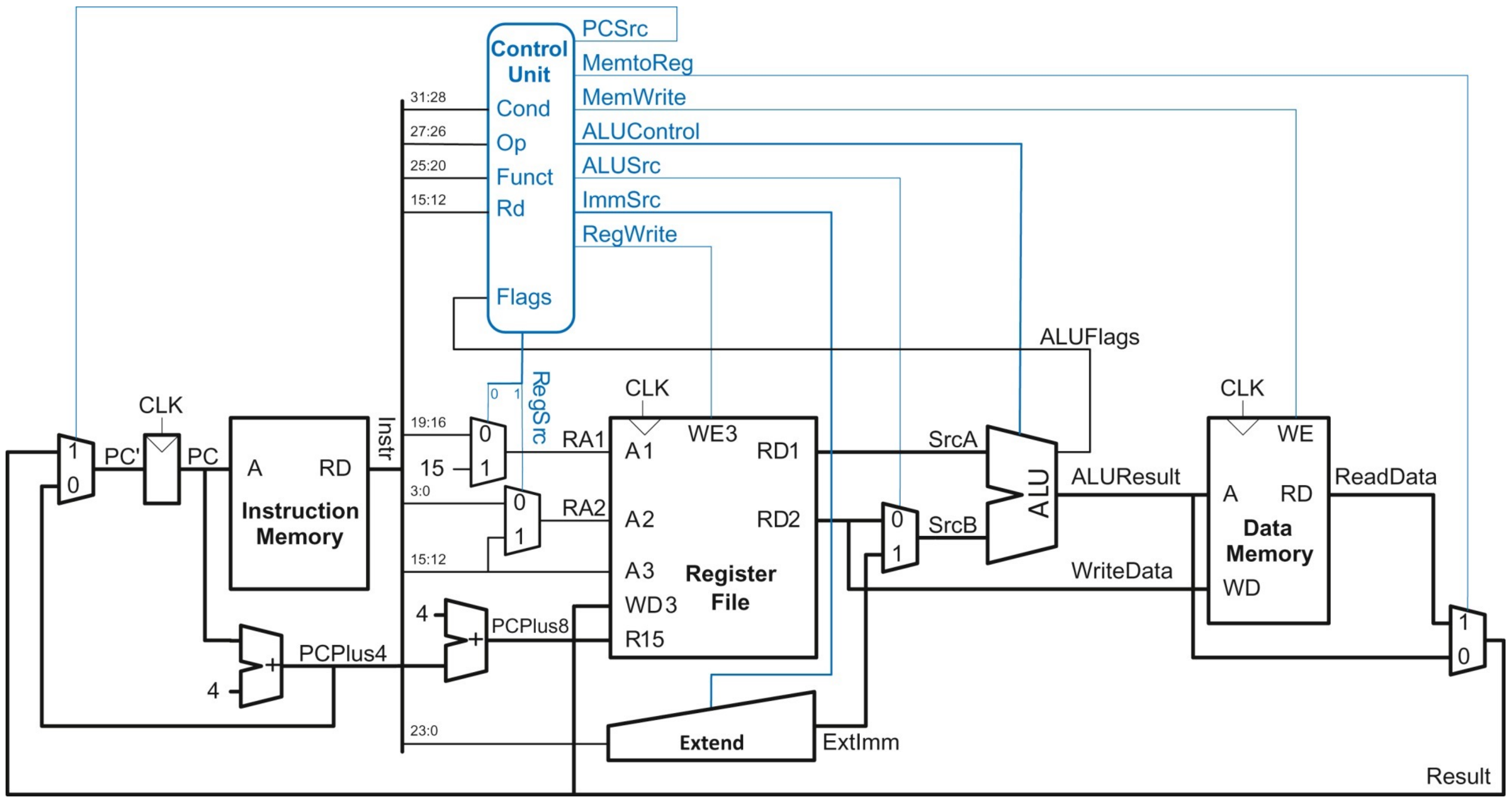
PCSrc	0
MemtoReg	0
MemWrite	0
ALUControl	11
ALUSrc	0
$ImmSrc_{0:1}$	XX
RegWrite	1
$RegSrc_{0:1}$	00

ALUControl	Function
00	ADD
01	SUB
10	AND
11	ORR

Processor Operation: ORR



Processor Operation: LDR



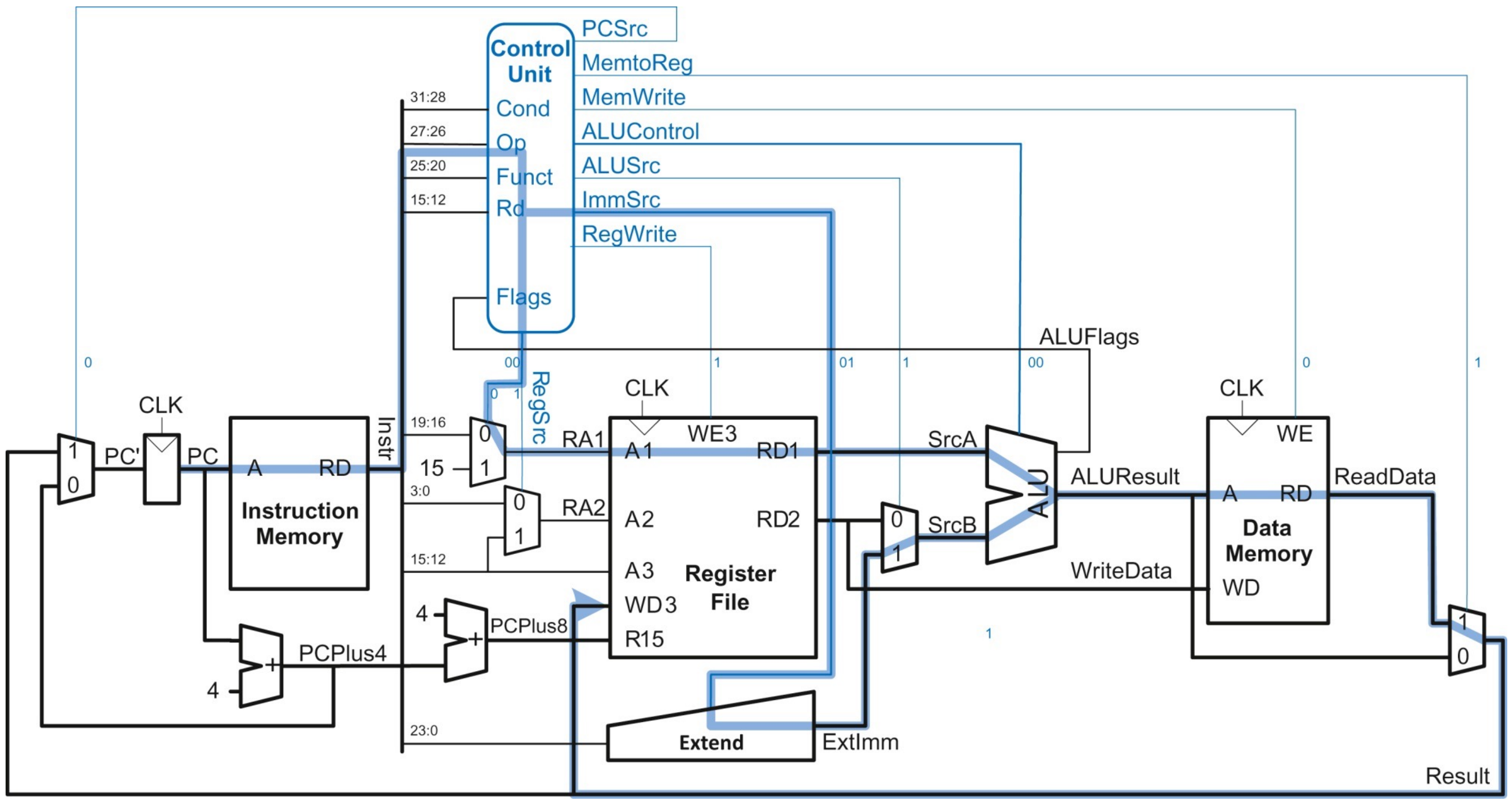
Processor Operation: LDR

PCSrc	0
MemtoReg	1
MemWrite	0
ALUControl	00
ALUSrc	1
$ImmSrc_{0:1}$	01
RegWrite	1
$RegSrc_{0:1}$	00

ALUControl	Function
00	ADD
01	SUB
10	AND
11	ORR

$ImmSrc_{1:0}$	ExtImm	Description
00	{24'b0, $Instr_{7:0}$}	Zero-extended *imm8*
01	{20'b0, $Instr_{11:0}$}	Zero-extended *imm12*
10	{6{$Instr_{23}$}, $Instr_{23:0}$}	Sign-extended *imm24*

Processor Operation: LDR



Whiteboard

Exercise: Generating PCSrc Signal

- PCSrc is 1 when
  - Destination register (Rd) is R15
  - RegW is 1 (ADD/SUB or LDR)
  - Instruction is a branch
- PCSrc = ((Rd == 15) & RegW) | Branch
  - Assuming the control unit generates a signal called Branch when opcode is 10 (B or BL)
- **Important:** Be careful to take conditional execution into account in the lab task and assignment!

Critical Path Analysis

- Each instruction in our CPU takes one clock cycle
- To determine the clock cycle time requires us to find the critical path
- Different instructions use different resources
  - LDR uses instruction and data memory
  - ADD does not use data memory
  - STR does not write anything back to the register file
- Which instruction is the slowest?
  - Let’s go back to the figures and find out!

Elements of Critical Path

Parameter	Description
t_{pcq_PC}	PC clock-to-Q delay
t_{mem}	Memory read
t_{dec}	Decoder propagation delay
t_{mux}	Multiplexer delay
t_{RFread}	Register file read
t_{ext}	Extension block delay
t_{ALU}	ALU delay
$t_{RFsetup}$	Set up RF for write (next cycle)

Critical Path: LDR

$$T_c = t_{pcq\_PC} + t_{mem} + t_{dec} + \max[t_{mux} + t_{RFread}, t_{ext} + t_{mux}] + t_{ALU} + t_{mem} + t_{mux} + t_{RFsetup}$$

- Memories & register files slower than combinational logic
  - Therefore, $t_{mux} + t_{RFread} >> t_{ext} + t_{mux}$

Final Equation

$$\boxed{T_c = t_{pcq\_PC} + 2t_{mem} + t_{dec} + t_{RFread} + t_{ALU} + 2t_{mux} + t_{RFsetup}}$$

Critical Path: DP-R

$T_c = t_{pcq\_PC} + t_{mem} + t_{dec} + t_{mux} + t_{RFread} + t_{mux} + t_{ALU} + t_{mux}$
$+ t_{RFsetup}$

Final Equation

$$\boxed{T_c = t_{pcq\_PC} + t_{mem} + t_{dec} + t_{RFread} + t_{ALU} + 3t_{mux} + t_{RFsetup}}$$

Critical Path Analysis

- Different instructions have different critical paths
  - LDR is the slowest instruction
  - DP-R and B have shorter critical paths because they do not need to access data memory
- Single-cycle processor is a synchronous sequential circuit
  - Clock period is constant and long enough to accommodate the slowest instruction
- The numerical values of different variables in the critical path equation depend on the specific technology

Single-Cycle Microarchitecture

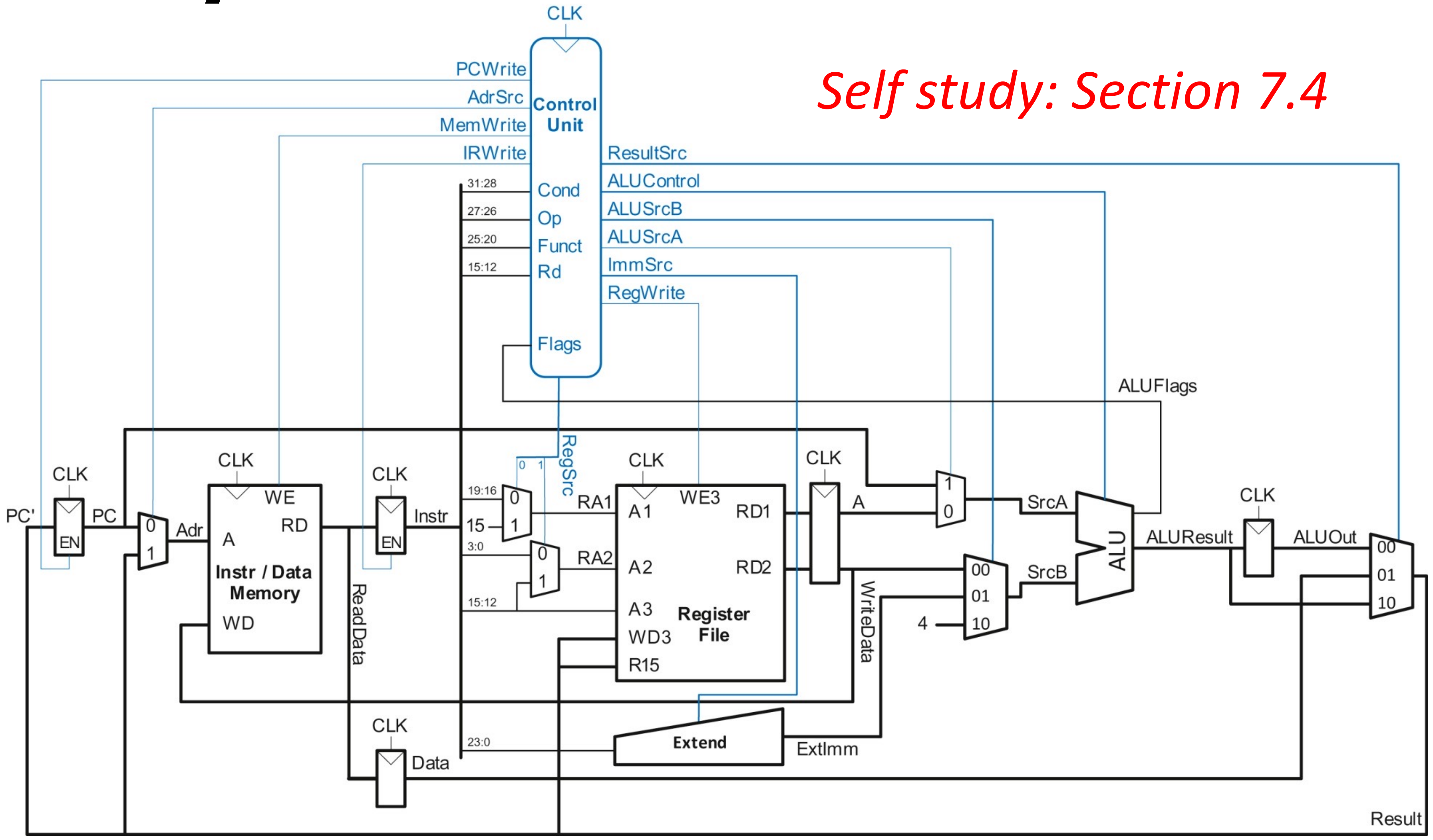
- Single-cycle microarchitecture
  - Execute the entire instruction in a single cycle
- Drawbacks
  - Separate memories for instructions and data
  - Cycle time is determined by the slowest instruction
  - Requires three adders (expensive)

Multicycle Microarchitecture

- Break the instruction into shorter steps
  - Access memory or use ALU in one step
  - Read instruction in one step and data in another step
  - Different instructions take different # steps or cycles
- Advantages
  - Some instructions finish faster than others
  - Needs one adder and one memory
- Drawbacks
  - Decoder complexity (decoder is an FSM)

Multicycle uarch

Self study: Section 7.4



Architectual State

- Modern systems run many applications at a time
- The operating system *interrupts* a program and runs another one, before restarting the interrupted program
- **Architectural state:** State needed to safely restart the program if it is interrupted by the operating system
- What state do we need to save and restore in ARM architecture?
  - 16 registers
  - Status register

Microarchitectural State

- Any other state in the CPU is called *microarchitectural* state or *non-architectural* state
  - This state is not visible to assembly programs
  - Programmers are unaware of microarchitectural state
- **Note:** Memory is part of the architectural state but typically we do not consider memory part of CPU
  - Memory and storage is part of the computer system

Pipelined Microarchitecture

- Multicycle microarchitecture
  - At any time, only one instruction is in execution
- Pipelined microarchitecture
- Pipelining divides the single-cycle CPU into stages
  - Each instruction goes through many stages
  - In each clock cycle, multiple instructions are active
  - One instruction can be in Fetch stage, another one can be in decode stage, ...
- Pipelining results in a shorter clock cycle time

Pipeline Stages

- What steps/stages do an LDR instruction goes through?
  - Fetch
  - Decode (Register Read)
  - Execute (ALU)
  - Memory Access
  - Writeback

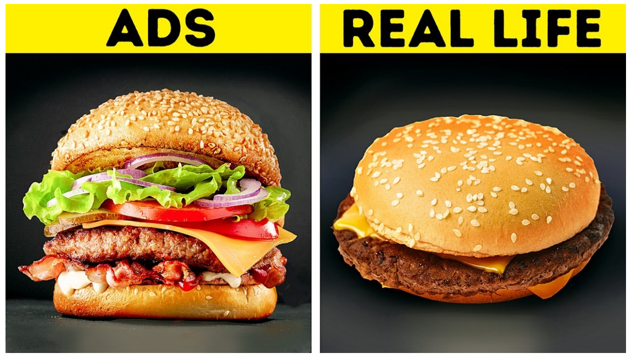
Pipeline Stages

- What steps/stages do an ADD instruction goes through?
  - Fetch
  - Decode (Register Read)
  - Execute (ALU)
  - ~~Memory Access~~
  - Writeback
- More details: We will return to pipelining details after a few lectures!

Performance Analysis

- We want the fastest (*best performing*) computer for a task
  - How should we measure and report performance?
- Which metric is *fair* for comparing two computers?
  - # Instructions
  - Clock frequency
  - Cycles per Instruction (**CPI**)
  - # Cores

Important to understand the true, gimmick-free measure of computer performance



# Instructions

- RISC computer
  - Many more simple instructions
  - Simple hardware means smaller clock cycle
- CISC computer
  - Small number of instructions
  - Complex hardware means larger clock cycle

Bottomline: *Number of instructions alone is not a good metric for quantifying the performance of application A*

Clock Frequency

- Consider the two scenarios
  - Computer A has a faster clock than computer B, but A executes many more instructions than B
  - Computer A has a faster clock than computer B, but takes multiple cycles to finish/execute a single instruction
- Is A faster than B?

Bottomline: *Clock frequency alone is not a good metric for quantifying the performance of application A*

CPI

- Cycles per instruction
  - Ratio of # cycles to # instructions
- A program has 10 instructions. Each instruction takes one cycle
  - Instructions = 10, Cycles = 10, CPI = 1
- A program has 10 instructions. Two out of 10 instructions take two cycles
  - Instructions = 10, Cycles = 14, CPI = 1.4
- The inverse of CPI is called IPC
  - Instructions per cycle
  - For the above examples, IPC is 1 and 0.7

CPI

- How can each instruction take multiple cycles?
  - Multi-cycle CPU
  - Memory accesses take more than one cycle
- On most computers, LDR takes variable # cycles because
  - Data may be present in faster (SRAM) memory called CPU cache
  - Or it may be present in main memory which is much slower

CPI

- Two computers A and B have the same CPI for a specific program. Is there performance equivalent?
  - We need to know the cycle time
  - We need to know the # instructions

Bottomline: *CPI alone is not a good metric for quantifying the performance of application A*

Execution Time

- The time it takes for a program to execute from start to finish is the only true measure of performance

$$\text{Execution time} = (\#\text{instructions})\left(\frac{cycles}{instruction}\right)\left(\frac{seconds}{cycle}\right)$$

- seconds per cycle = cycle time
- Execution time is measured in seconds
- Golden metric for quantifying computer performance!

Execution Time

$$\text{Execution time} = (\#\text{instructions})\left(\frac{cycles}{instruction}\right)\left(\frac{seconds}{cycle}\right)$$

# instructions

- Depends on the ISA, skill of programmer, compiler, algorithm

cycles per instruction

- Depends on the microarchitecture esp. memory system

seconds per cycle

- critical path, circuit technology, type of adders, gate-level details

Exercise: Perf Analysis

- Find the time it takes to execute a program with 100 billion instructions on a single-cycle CPU in 16 nm CMOS manufacturing process. See the table for delays of logic elements.

Parameter	Delay (ps)
t_{pcq_PC}	40
t_{mem}	200
t_{dec}	70
t_{mux}	25
t_{RFread}	100
t_{ALU}	120
$t_{RFsetup}$	60

$$\boxed{T_c = t_{pcq\_PC} + 2t_{mem} + t_{dec} + t_{RFread} + t_{ALU} + 2t_{mux} + t_{RFsetup}}$$

Measurement Methodology

- The good practice: Take a program of interest and measure its execution time
- The better practice: Take a collection of programs like the programs of interest, and measure their performance
  - You do not have the program yet
  - Some one else is measuring the performance independently
- This collection of programs is called a *benchmark suite*
  - Dhrystone and CoreMark (embedded systems)
  - SPEC (Standard Performance Evaluation Corporation)
  - SPEC is standard suite for high-performance processors

For Loop in C

C code:
```
int i;
int sum = 0;

for (i = 0; i < 10; i = i + 1) {
        sum = sum + i;
}
```

- The variable “i” is called the loop index
- i = 0 : index initialization
- i < 10 : loop termination condition
- i = i + 1 : loop advancement

For Loop: C to Assembly

C code:

```c
int i;
int sum = 0;

for (i = 0; i < 10; i = i + 1) {
      sum = sum + i;
}
```

ARM Assembly code

; R0 = i, R1 = sum

```
        MOV     R1,     #0
        MOV     R0,     #0
FOR
        CMP     R0,     #10
        BGE     DONE
        ADD     R1,     R1,     R0
        ADD     R0,     R0,     #1
        B       FOR
DONE
```

check termination condition to break out of the loop if condition is met

keep iterating by branching back

For Loop: Perf Analysis

ARM Assembly code

; R0 = i, R1 = sum

```
        MOV     R1,     #0
        MOV     R0,     #0
FOR
        CMP     R0,     #10
        BGE     DONE
        ADD     R1,     R1,     R0
        ADD     R0,     R0,     #1
        B       FOR
DONE
```

- The clock cycle time T_c is 840 ps
- Find the time it takes to execute the for loop
  - # instructions = ?
  - CPI = 1
  - Execution time = ?

Alternative For Loop

```
        MOV   R1,   #0
        MOV   R0,   #0
COND
        CMP   R0,   #10
        BLT   LOOP
        B     DONE
LOOP
        ADD   R1,   R1,   R0
        ADD   R0,   R0,   #1
        B     COND
DONE
```

- We can implement the for loop in a different way
- Find the time it takes to execute the for loop again
  - # instructions = ?
  - Execution time = ?

Bottom line: Execution time depends on how we write code and microarchitecture details

While Loop in C

C code:

```
int pow = 1;
int x = 0;

while (pow != 128) {
        pow = pow * 2;
        x = x + 1;
}
```

- For loop: iterate N times
- While loop: Iterate until a condition is not met

While Loop in C

C code:

```c
int pow = 1;
int x = 0;

while (pow != 128) {
        pow = pow * 2;
        x = x + 1;
}
```

ARM Assembly code

; R0 = pow, R1 = x

```
        MOV     R0,     #1
        MOV     R1,     #0
WHILE
        CMP     R0,     #128
        BEQ     DONE
        LSL     R0,     R0,     #1
        ADD     R1,     R1,     #1
        B       WHILE
DONE
```

Exercise

- The clock cycle time T_c is 840 ps
- Find the time it takes to execute the While loop
  - # instructions = ?
  - CPI = 1
  - Execution time = ?

Shift Instructions

- Shift the value in a register left or right, drop bits off the end
    - Logical shift left (LSL)
    - Logical shift right (LSR)
    - Arithmetic shift right (ASR)
    - Rotate right (ROR)
- Logical shift: shifts the number to the left or right and fills the empty slots with zero
- Arithmetic shift: on right shifts fill the most significant bits with zero
- Rotate: rotates number in a circle such that empty spots are filled with bits shifted off the other end

Example: Shift Operations

- **Immediate** shift amount (5-bit immediate)
- Shift amount: 0-31

Source register

R5	1111 1111	0001 1100	0001 0000	1110 0111

Assembly Code		Result			
`LSL R0, R5, #7`	R0	1000 1110	0000 1000	0111 0011	1000 0000
`LSR R1, R5, #17`	R1	0000 0000	0000 0000	0111 1111	1000 1110
`ASR R2, R5, #3`	R2	1111 1111	1110 0011	1000 0010	0001 1100
`ROR R3, R5, #21`	R3	1110 0000	1000 0111	0011 1111	1111 1000

Shifts: Machine Representation

DP-R

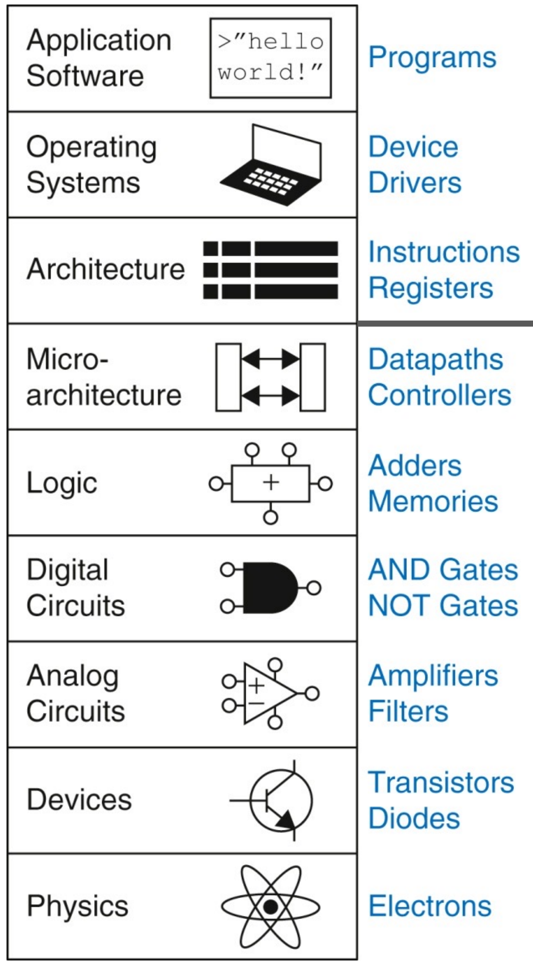
31:28	27:26	25	24:21	20	19:16	15:12	11:4								3:0
cond	00	0	cmd	S	Rn	Rd	0	0	0	0	0	0	0	0	Rm

Shift Instructions

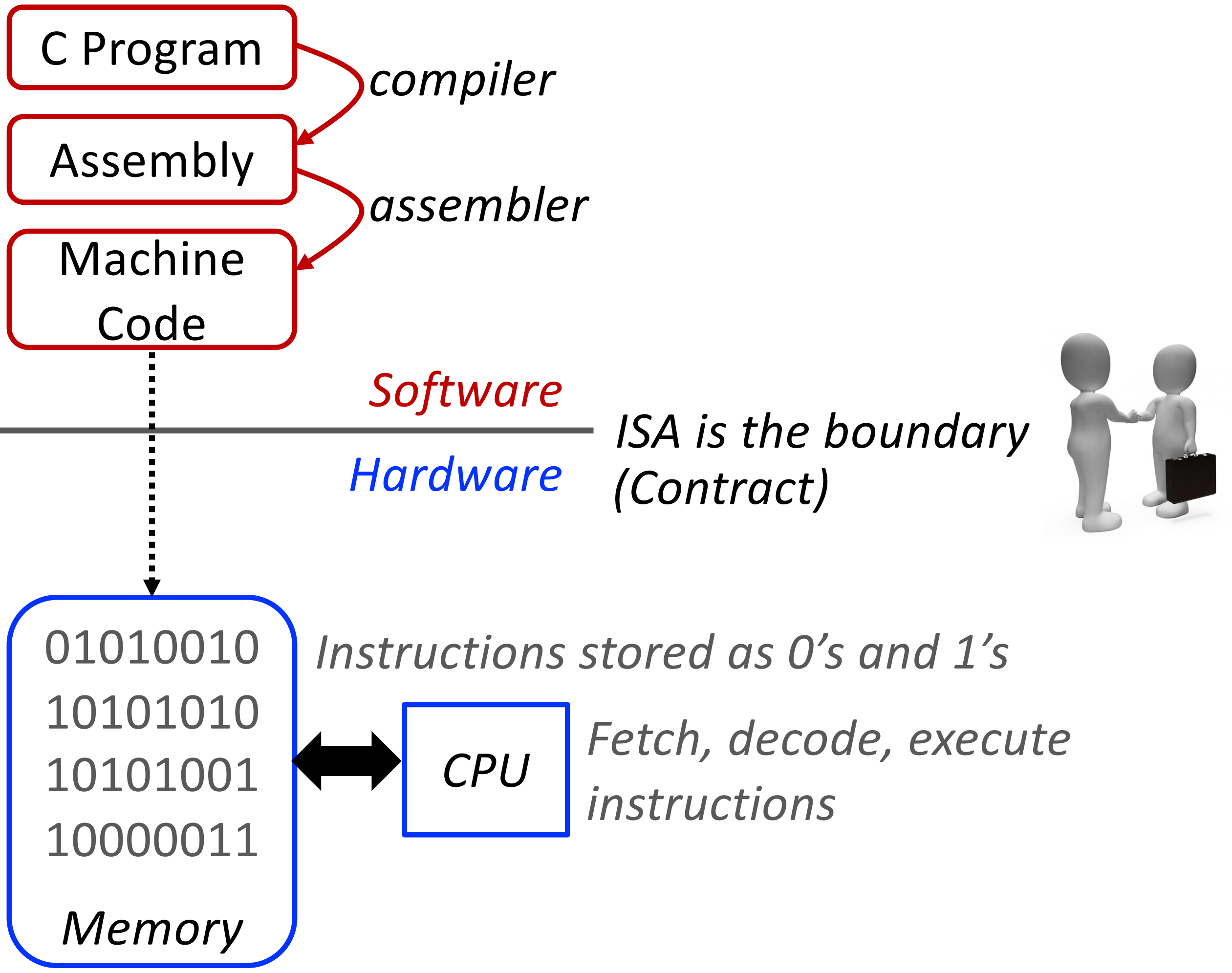
31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0
cond	00	0	cmd	S	Rn	Rd	shamt5	sh	0	Rm

- cmd = 1101
- sh = 00 (LSL), 01 (LSR), 10 (ASR), 11 (ROR)
- Rn = 0
- shamt5 = 5-bit shift amount

Big Picture



Application Software
>"hello world!"
Programs
Operating Systems
Device Drivers
Architecture
Instructions Registers
Micro-architecture
Datapaths Controllers
Logic
Adders Memories
Digital Circuits
AND Gates NOT Gates
Analog Circuits
Amplifiers Filters
Devices
Transistors Diodes
Physics
Electrons



C Program
compiler
Assembly
assembler
Machine Code
Software
Hardware
ISA is the boundary (Contract)
01010010
10101010
10101001
10000011
Memory
Instructions stored as 0's and 1's
CPU
Fetch, decode, execute instructions

Happy Teaching Break!

Remember the Assignment (30%)

Keep the transformation hierarchy fresh in your memory!

See you in two weeks ☺