The MIPS Processor

Hakim Weatherspoon

CS 3410

Computer Science

Cornell University

The slides are the product of many rounds of teaching CS 3410 by Professors Weatherspoon, Bala, Bracy, and Sirer.

Announcements

Project Partner finding assignment on CMS

No official office hours over break

Announcements

Make sure to go to ***your*** Lab Section this week

Lab3 due in class this week (it is ***not*** homework)

Proj1: Completed Proj1 due ***this*** Friday, Feb 16th, ***before*** winter break

Note, a **Design Document** is due when you submit Proj1 final circuit

Work **alone**

Save your work!

- ***Save often***. Verify file is non-zero. Periodically save to Dropbox, email.
- Beware of MacOSX 10.5 (leopard) and 10.6 (snow-leopard)

C programming assignment is out

Due Tuesday, February 27th

Office Hours for help

Work **alone**

Work alone, **BUT** use your resources

- Lab Section, Piazza.com, Office Hours
- Class notes, book, Sections, CSUGLab

Announcements

Check online syllabus/schedule

- http://www.cs.cornell.edu/Courses/CS3410/2018sp/schedule
- Slides and Reading for lectures
- Office Hours
- ***Pictures of all TAs***

- **Dates to keep in Mind**
  - **Prelims: Thur Mar 15th and Thur May 3rd**
  - ***Proj 1: Due this Friday, Feb 16th before Winter break***
  - Proj3: Due before Spring break
  - Final Project: Due when final would be May 15th

Schedule is subject to change

Collaboration, Late, Re-grading Policies

"White Board" Collaboration Policy

- Can discuss approach together on a "white board"
- Leave, watch a movie such as Stranger Things, then write up solution independently
- Do not copy solutions

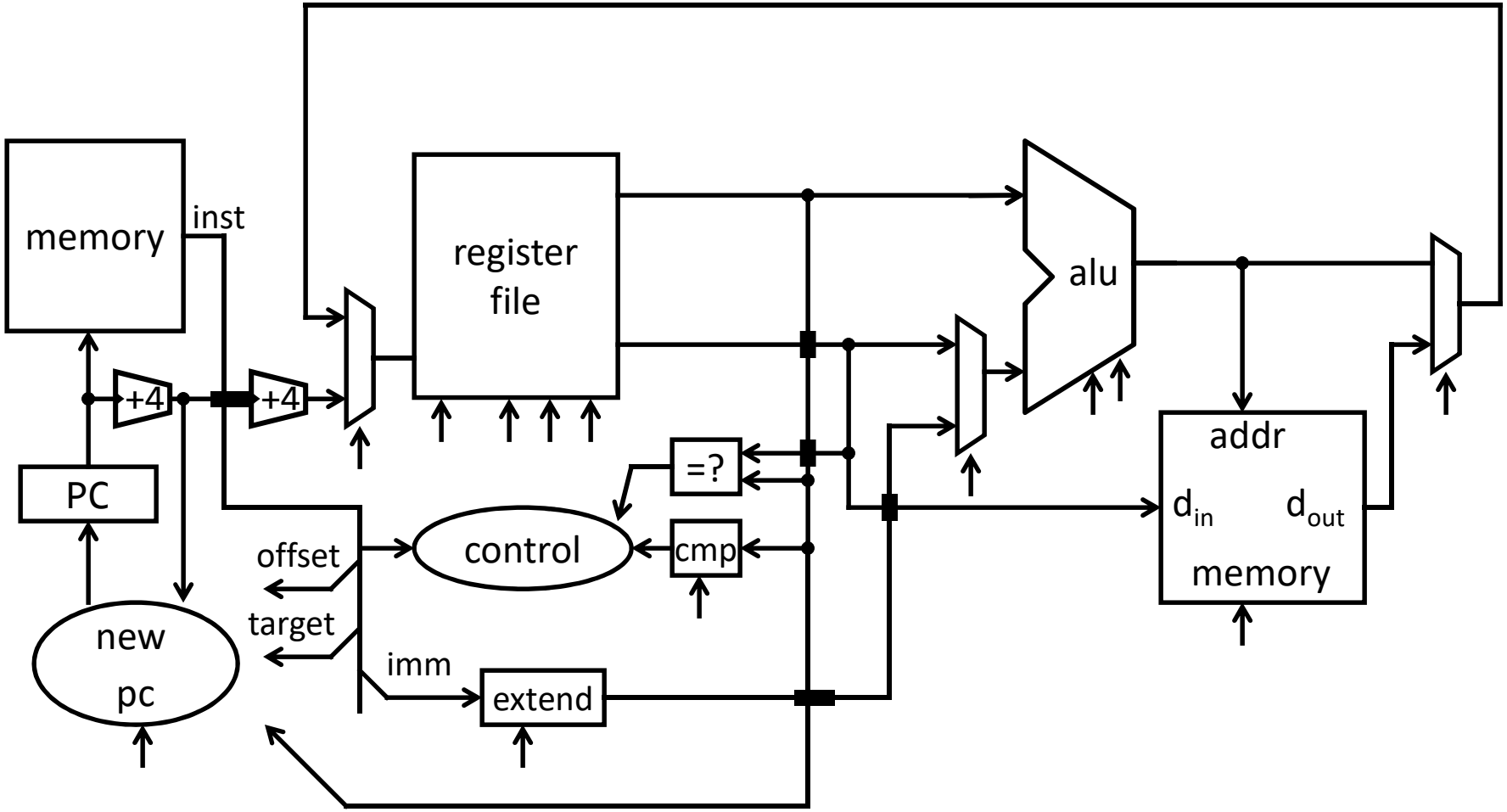
Late Policy

- Each person has a total of ***four*** "slip days"
- Max of ***two*** slip days for any individual assignment
- Slip days deducted first for *any* late assignment, cannot selectively apply slip days
- For projects, slip days are deducted from all partners
- ***25%*** deducted per day late after slip days are exhausted

Regrade policy

- Submit written request within a week of receiving score

Big Picture: Building a Processor



A Single cycle processor

Goal for the next 2 lectures

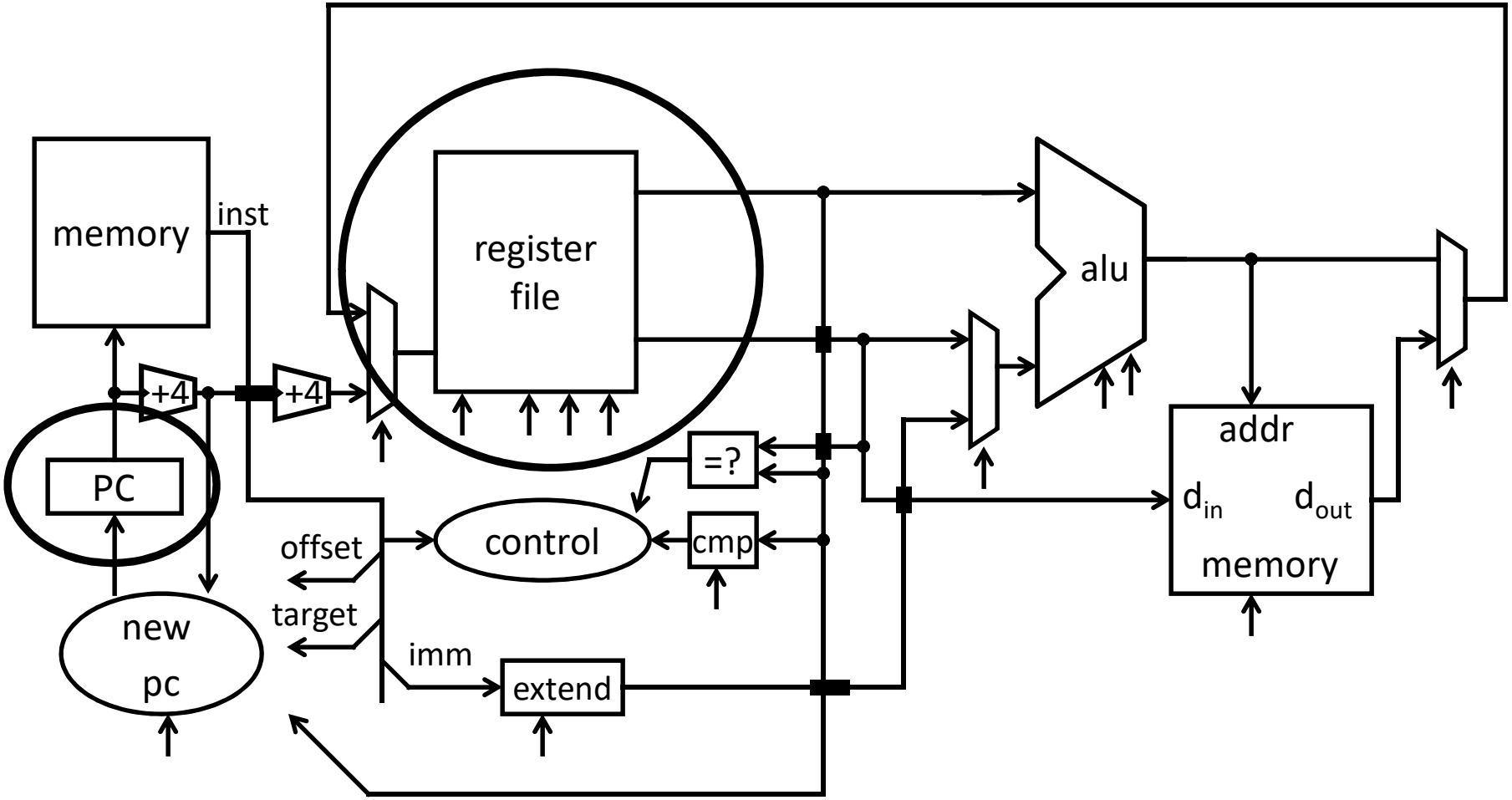
Understanding the basics of a processor

We now have the technology to build a CPU!

Putting it all together:

- Arithmetic Logic Unit (ALU)
- Register File
- Memory
  - SRAM: cache
  - DRAM: main memory
- MIPS Instructions & how they are executed

MIPS Register File

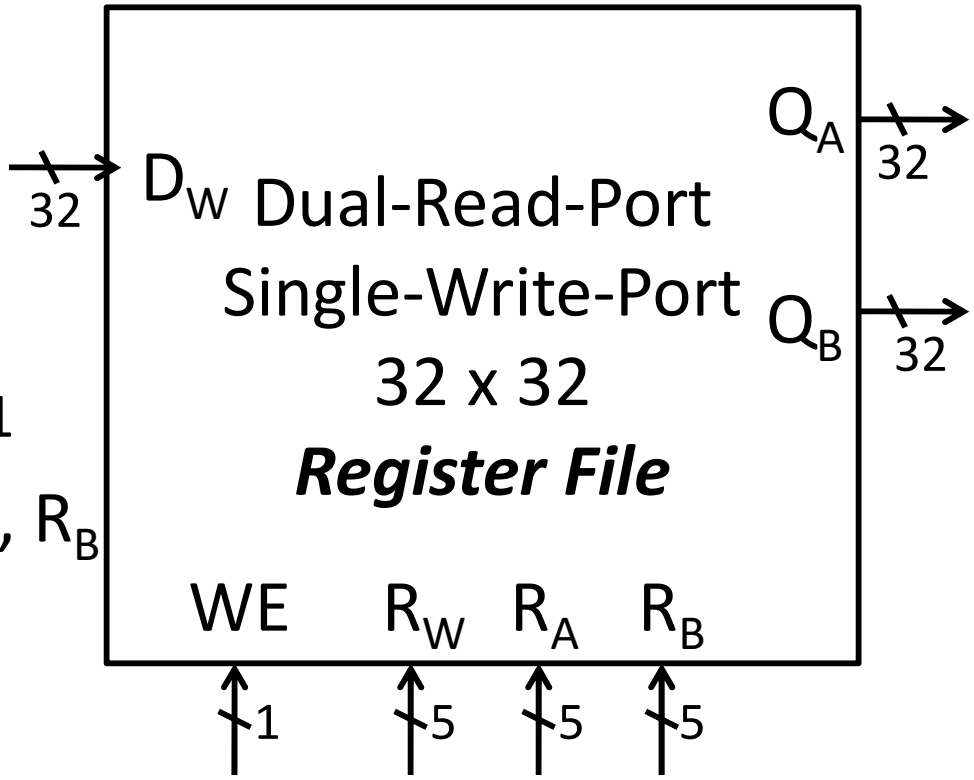


A Single cycle processor

MIPS Register File

MIPS register file

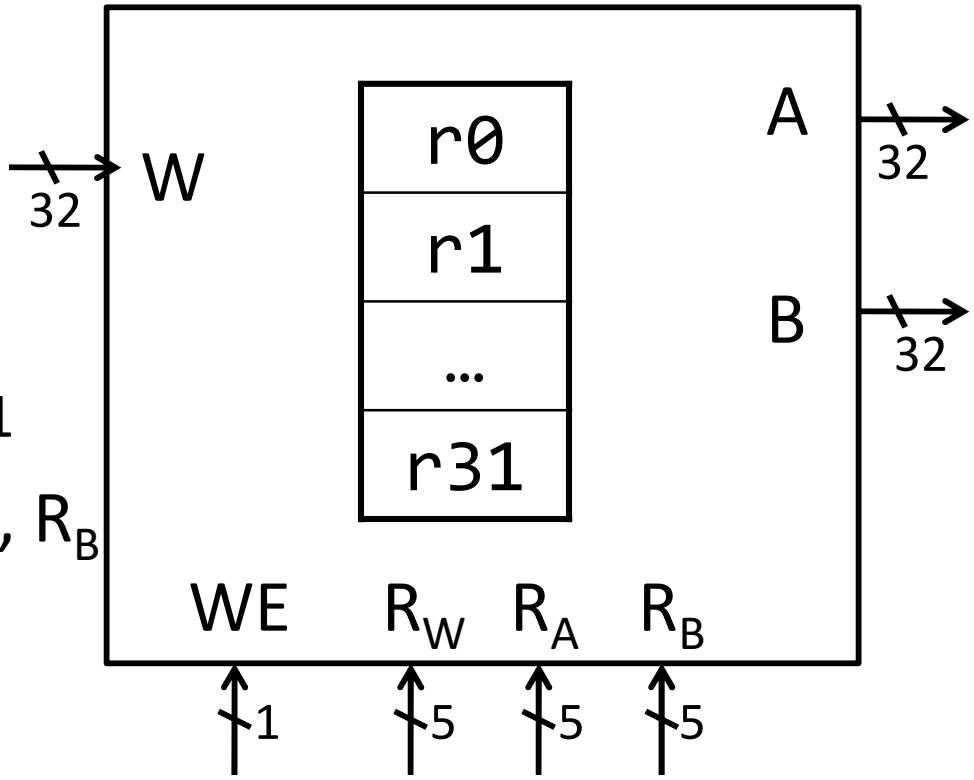
- 32 registers, 32-bits each
- r0 wired to zero
- Write port indexed via R_W
  - on falling edge when WE=1
- Read ports indexed via R_A, R_B



MIPS Register File

MIPS register file

- 32 registers, 32-bits each
- r0 wired to zero
- Write port indexed via R_W
  - on falling edge when WE=1
- Read ports indexed via R_A, R_B

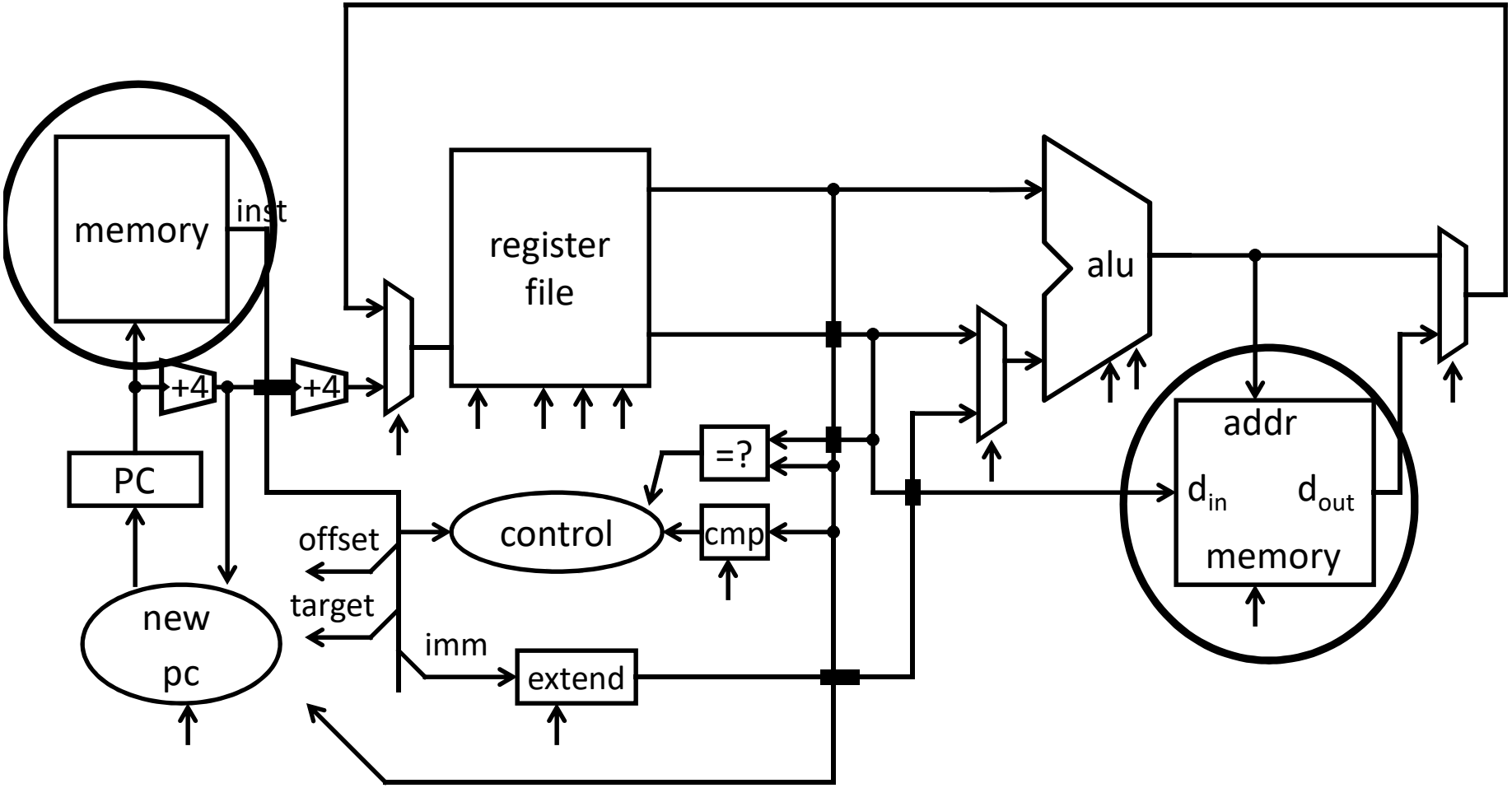


MIPS register file

- Numbered from 0 to 31.
- Can be referred by number: $0, $1, $2, … $31
- Convention, each register also has a name:
  - $16 - $23 → $s0 - $s7, $8 - $15 → $t0 - $t7

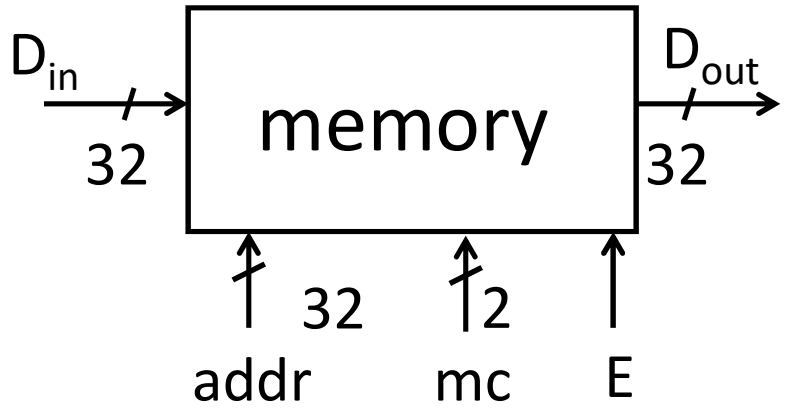
  [P&H p105]

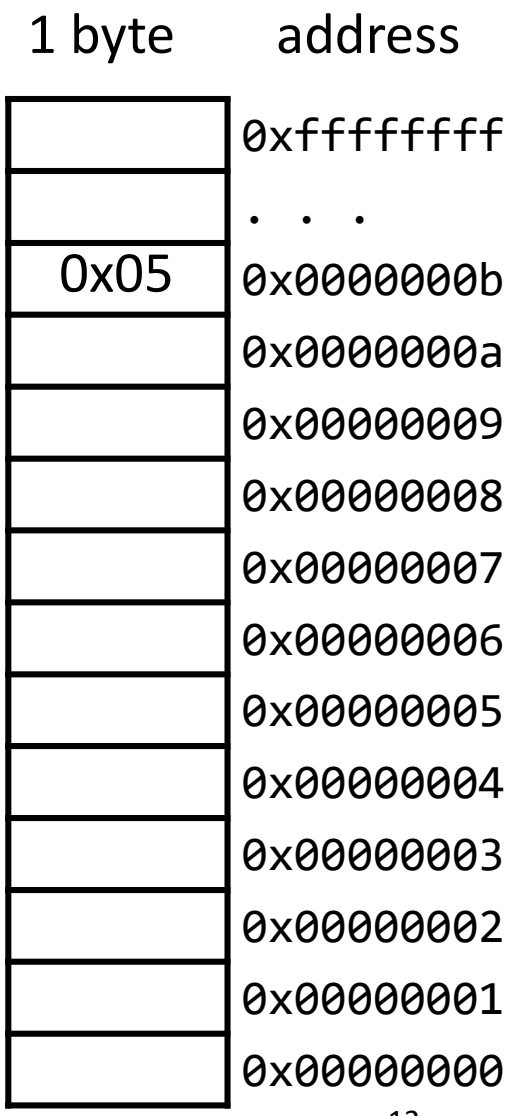
MIPS Memory



A Single cycle processor

MIPS Memory





- 32-bit address
- 32-bit data (but byte addressed)
- Enable + 2 bit memory control (mc)

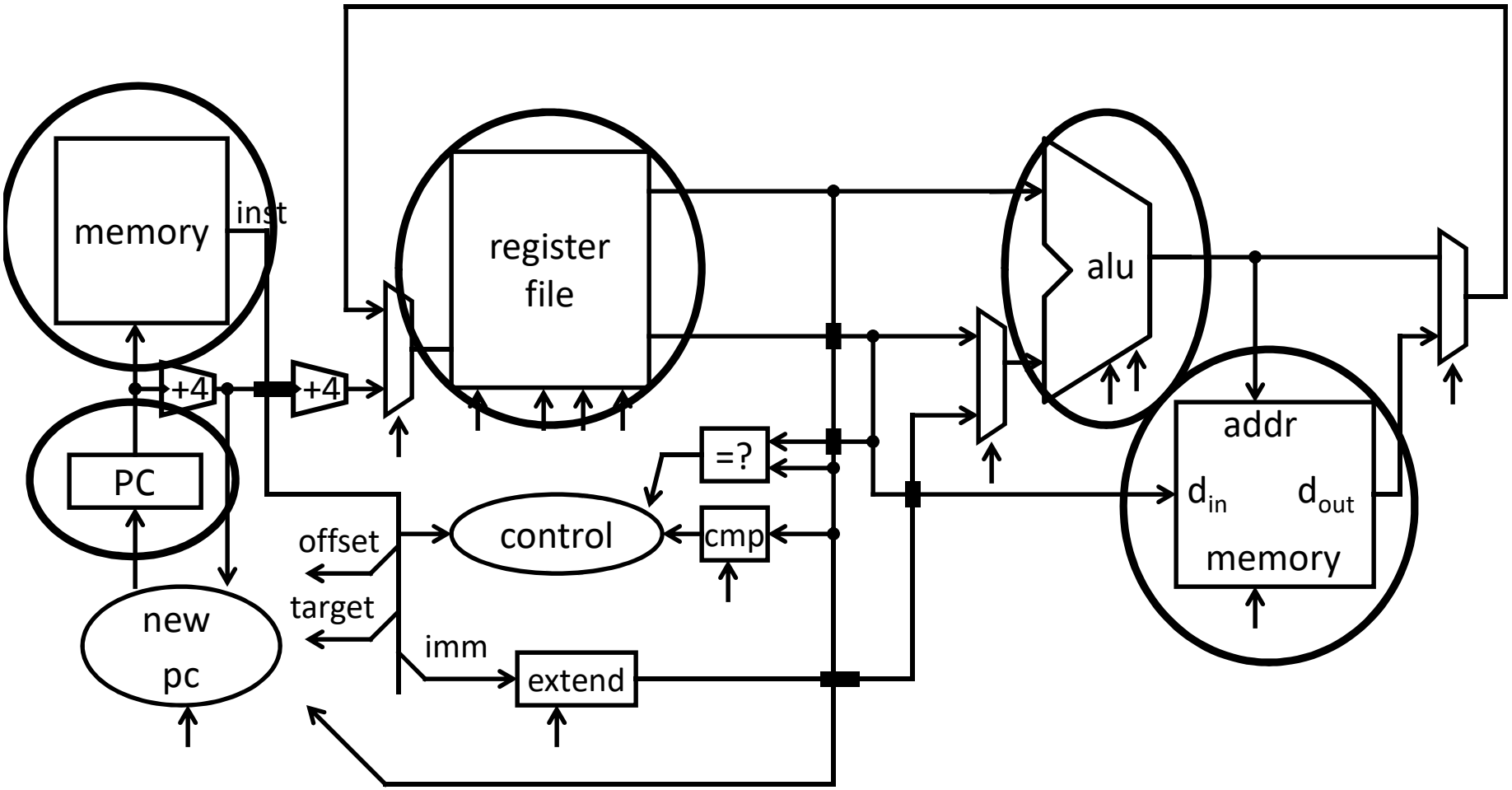
  00: read word (4 byte aligned)

  01: write byte

  10: write halfword (2 byte aligned)

  11: write word (4 byte aligned)

Putting it all together: Basic Processor



A Single cycle processor

To make a computer

Need a program

- Stored program computer

Architectures

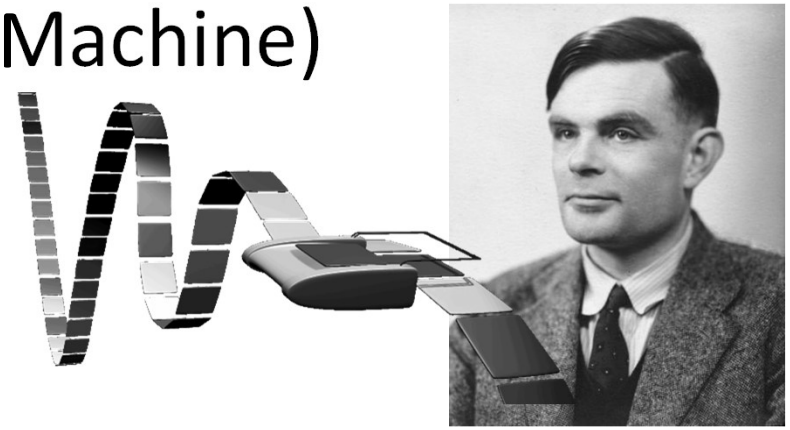
- von Neumann architecture
- Harvard (modified) architecture

To make a computer

Need a program

Stored program computer

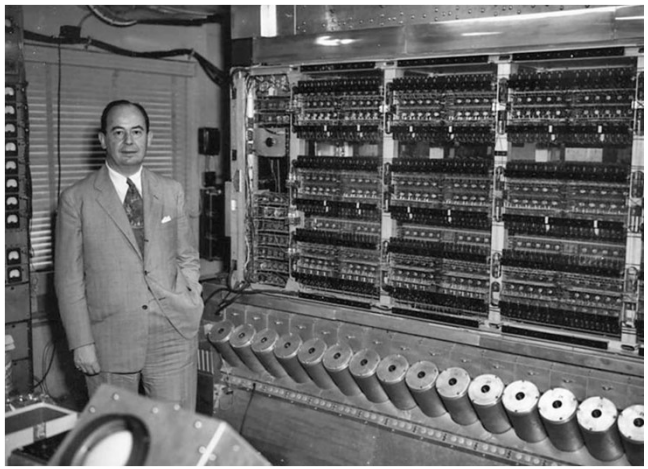
(a Universal Turing Machine)



Architectures

von Neumann architecture

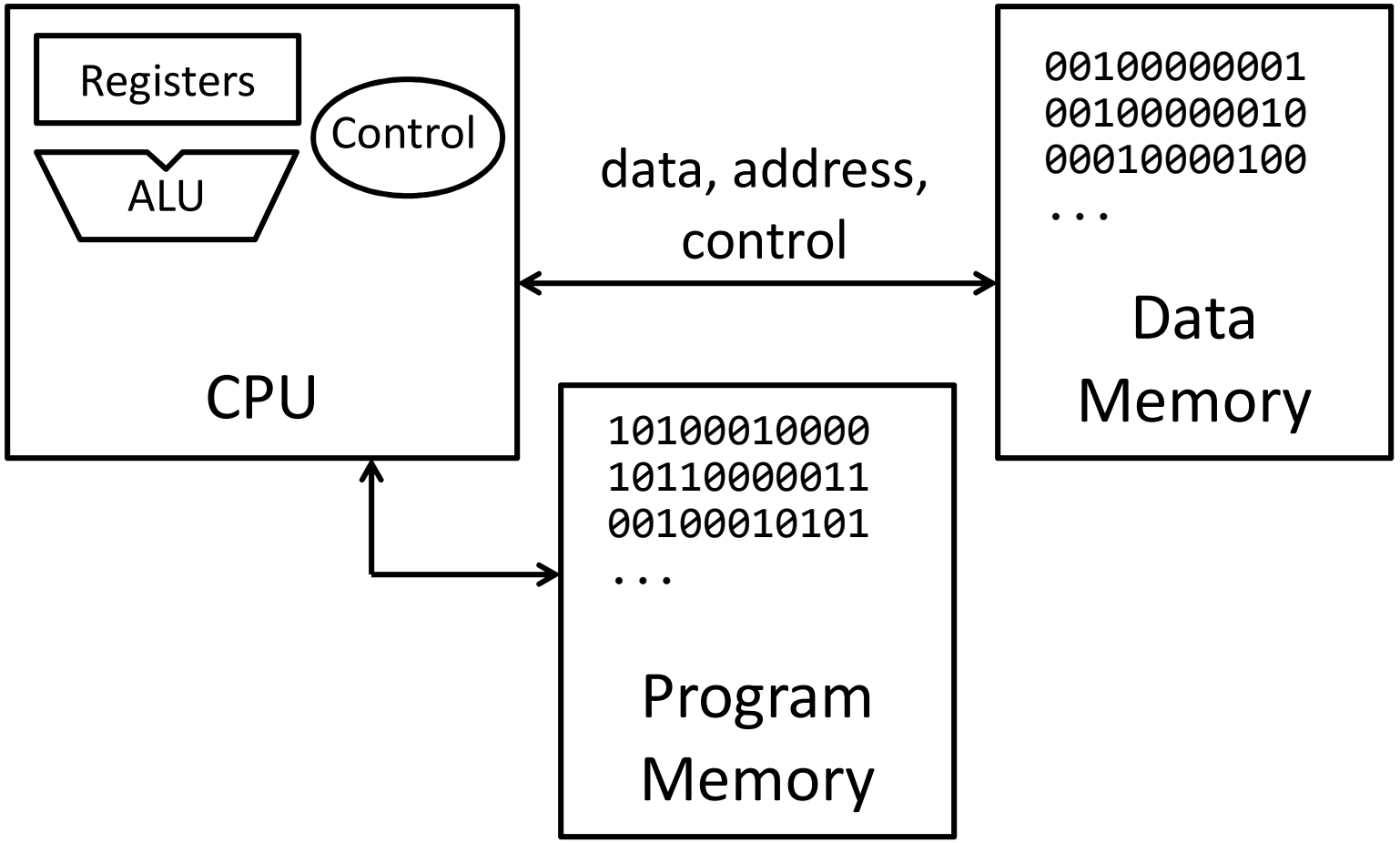
Harvard (modified) architecture



Putting it all together: Basic Processor

A MIPS CPU with a (modified) Harvard architecture

- Modified: instructions & data in common address space, separate instr/data caches can be accessed in parallel



Takeaway

A processor executes instructions

- Processor has some internal state in storage elements (registers)

A memory holds instructions and data

- (modified) Harvard architecture: separate insts and data
- von Neumann architecture: combined inst and data

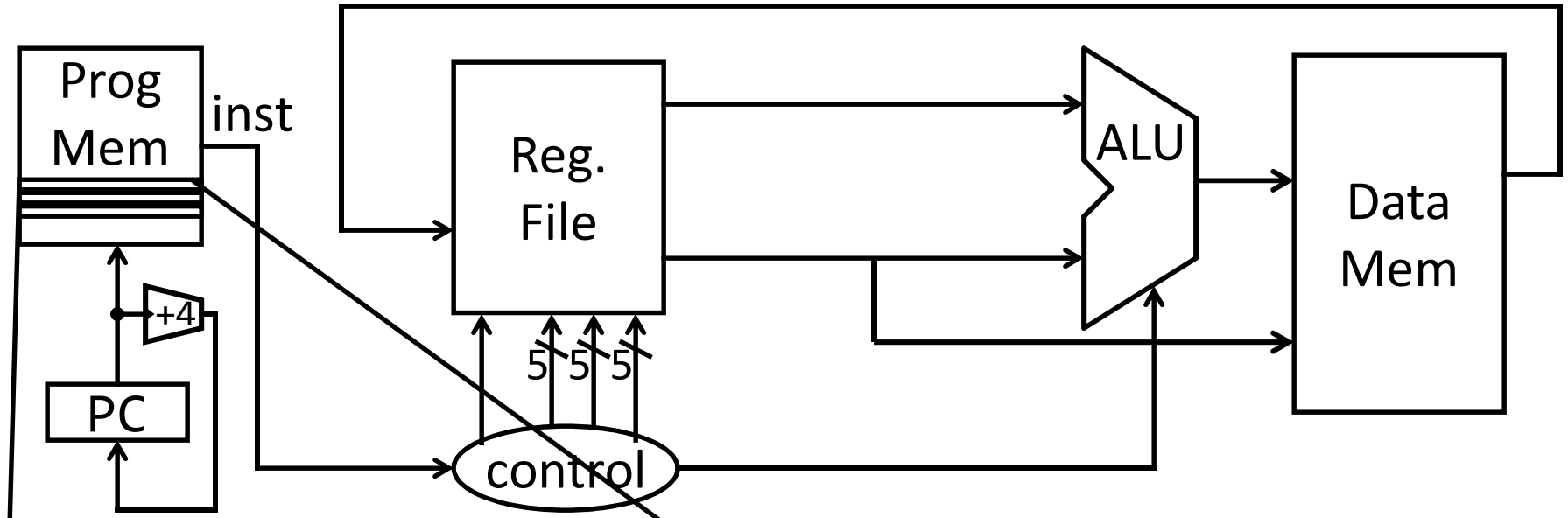
A bus connects the two

We now have enough building blocks to build machines that can perform non-trivial computational tasks

Next Goal

How to program and execute instructions on a MIPS processor?

Instruction Processing



Instructions:

stored in memory, encoded in binary

```
00100000000001000000000000001010
00100000000000010000000000000000
00000000001000100001100000101010
```

A basic processor

- fetches
- decodes
- executes

one instruction at a time

Levels of Interpretation: Instructions

```
for (i = 0; i < 10; i++)
        printf("go cucs");
```

High Level Language

- C, Java, Python, ADA, …
- Loops, control flow, variables

```
main: addi r2, r0, 10
      addi r1, r0, 0
 loop: slt r3, r1, r2
      ...
```

Assembly Language

- No symbols (except labels)
- One operation per statement
- "human readable machine language"

```
op=addi  r0    r2                    10
001000 00000 00010 0000000000001010
00100000000000010000000000000000
00000000001000100001100000101010
```

Machine Language

- Binary-encoded assembly
- Labels become addresses
- **The language of the CPU**

Instruction Set Architecture

ALU, Control, Register File, …

Machine Implementation (Microarchitecture)

Instruction Set Architecture (ISA)

Different CPU architectures specify different instructions

Two classes of ISAs

- Reduced Instruction Set Computers (RISC)
  IBM Power PC, Sun Sparc, MIPS, Alpha
- Complex Instruction Set Computers (CISC)
  Intel x86, PDP-11, VAX

Another ISA classification: Load/Store Architecture

- Data must be in registers to be operated on
  For example: array[x] = array[y] + array[z]
  1 add ? OR 2 loads, an add, and a store ?
- Keeps HW simple → many RISC ISAs are load/store

Takeaway

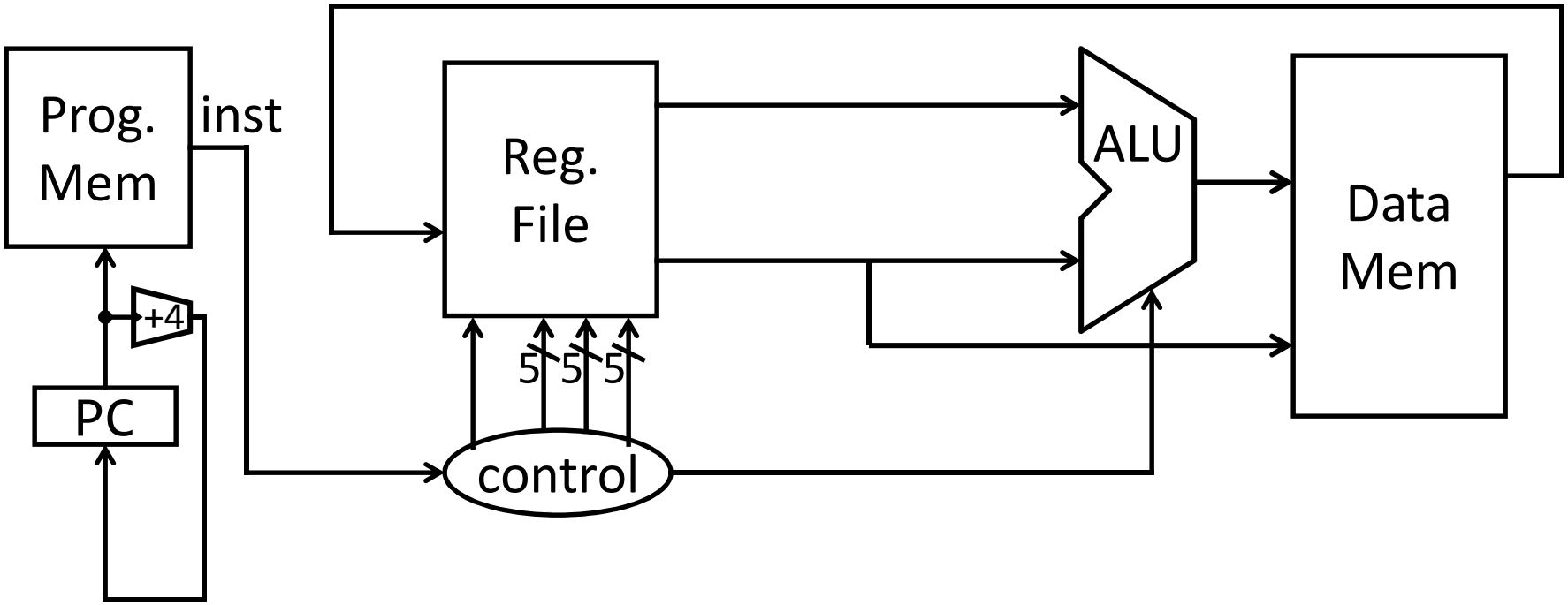
A MIPS processor and ISA (instruction set architecture) is an example a Reduced Instruction Set Computers (RISC) where simplicity is key, thus enabling us to build it!!

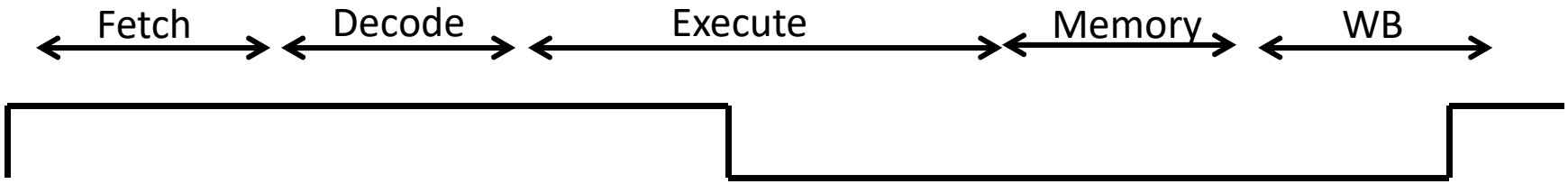
Next Goal

How are instructions executed?

What is the general datapath to execute an instruction?

Five Stages of MIPS Datapath





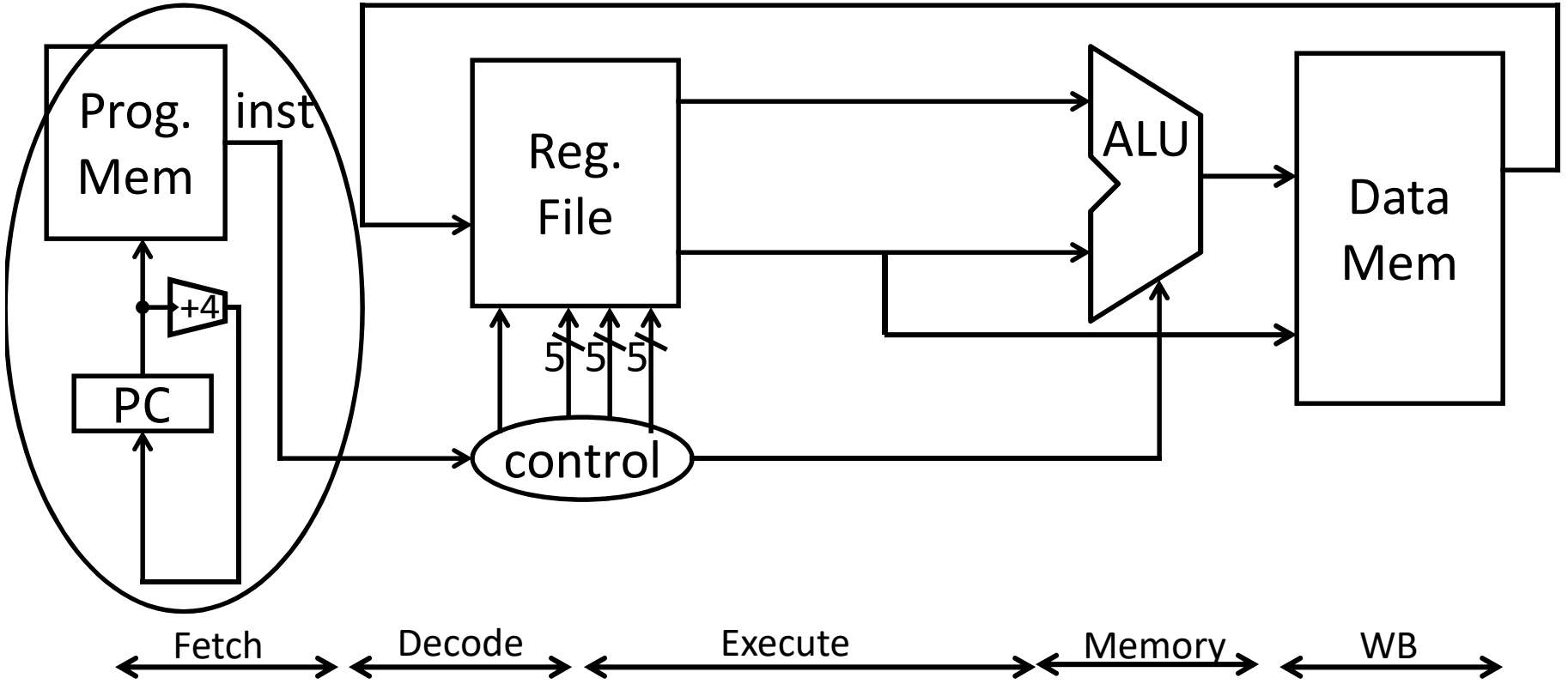
A Single cycle processor – this diagram is not 100% spatial

Five Stages of MIPS datapath

Basic CPU execution loop

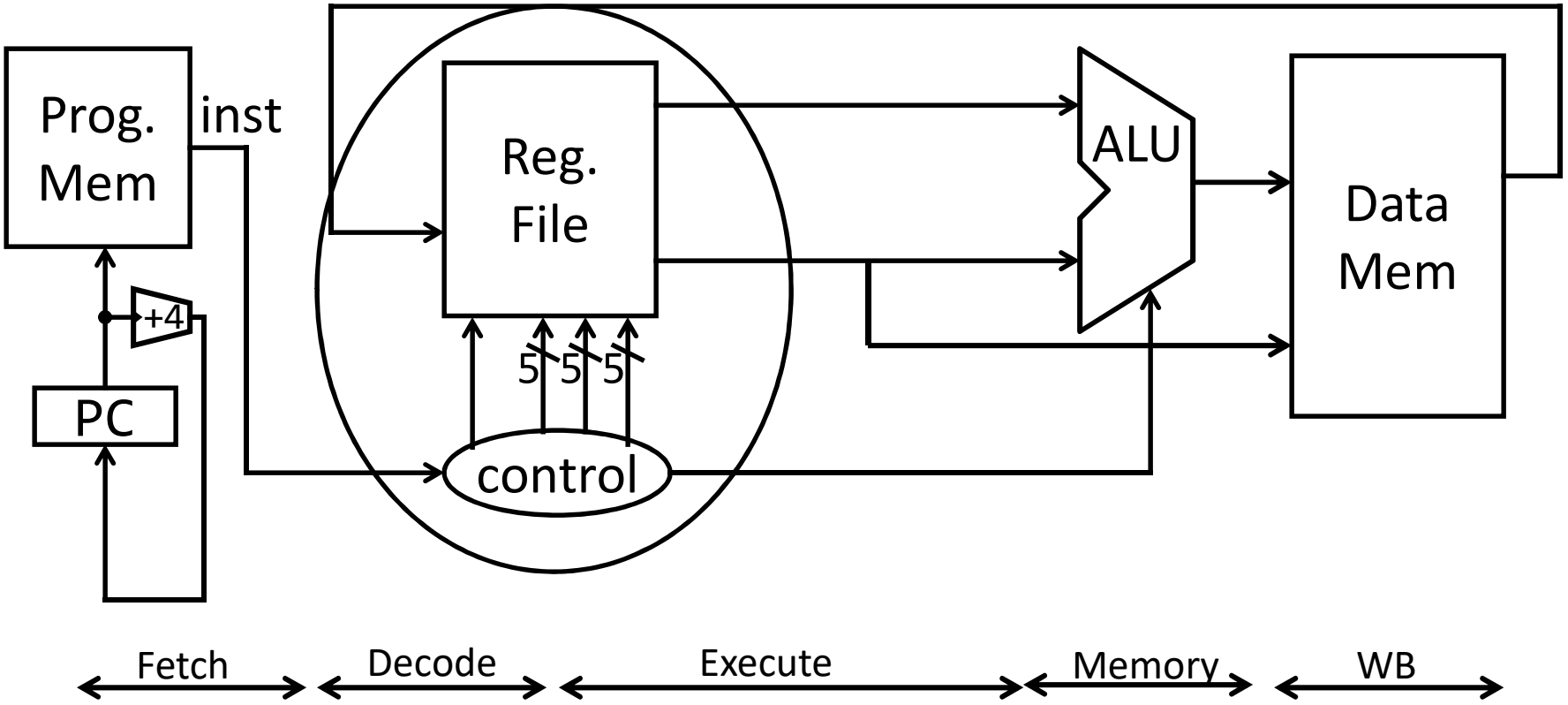
1. Instruction Fetch
2. Instruction Decode
3. Execution (ALU)
4. Memory Access
5. Register Writeback

Stage 1: Instruction Fetch



- Fetch 32-bit instruction from memory
- Increment PC = PC + 4

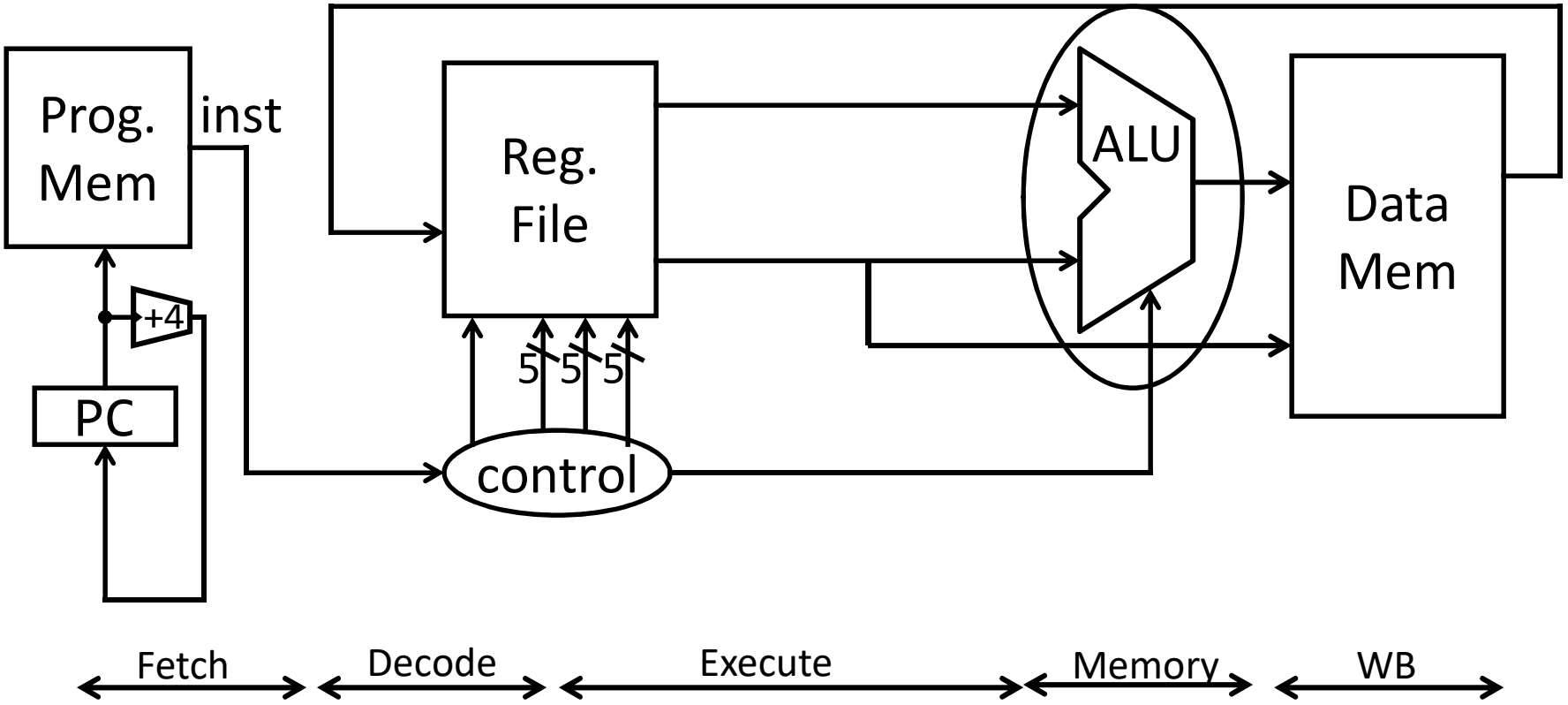
Stage 2: Instruction Decode



- Gather data from the instruction
- Read opcode; determine instruction type, field lengths
- Read in data from register file

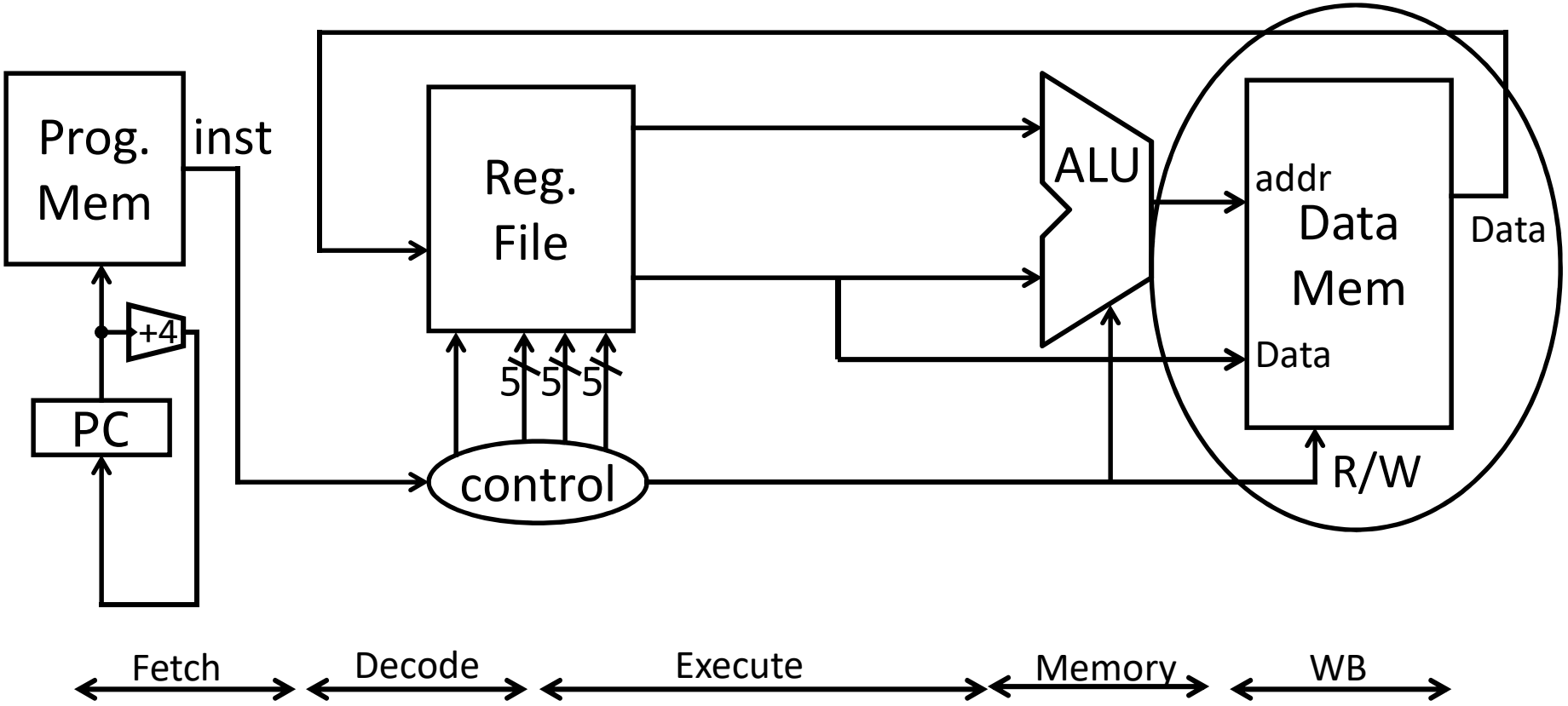
  (0, 1, or 2 reads for `jump`, `addi`, or `add`, respectively)

Stage 3: Execution (ALU)



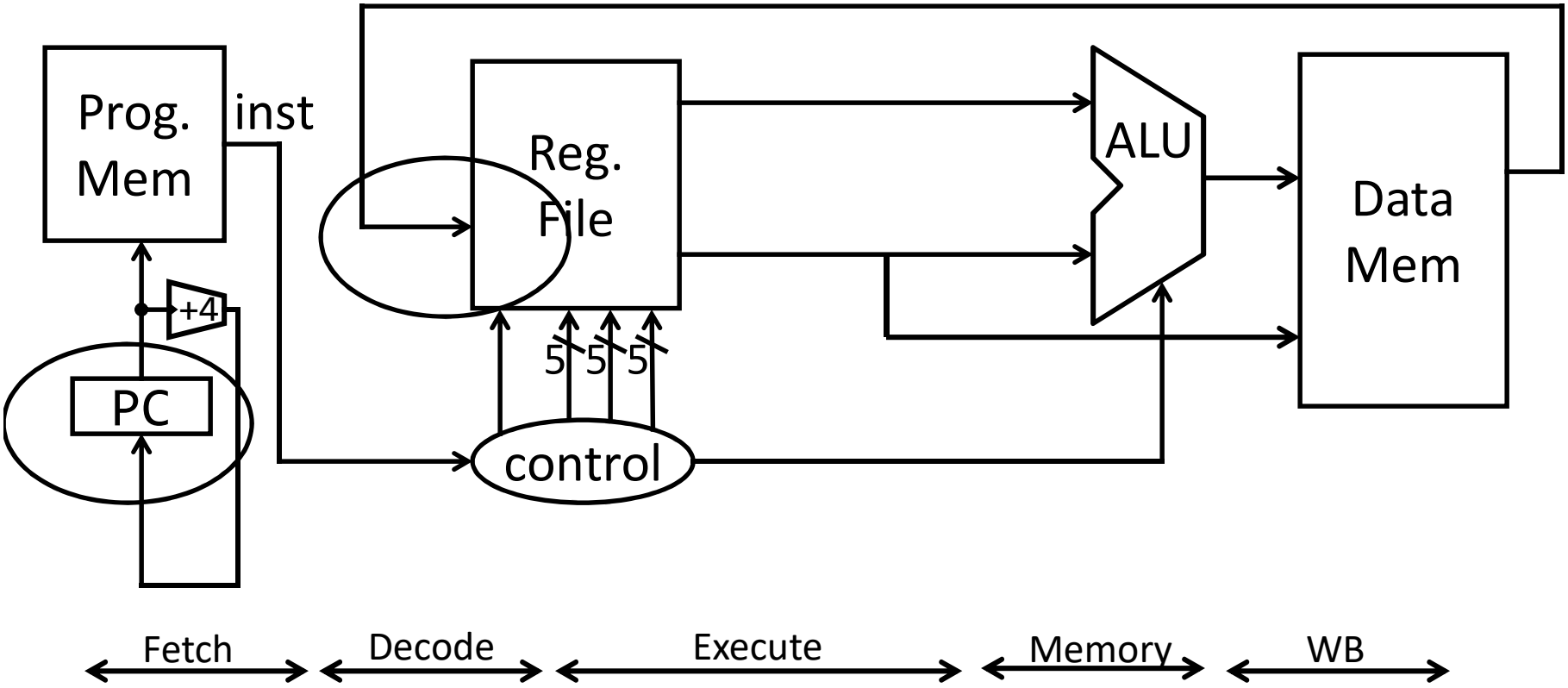
- Useful work done here (+, -, *, /), shift, logic operation, comparison (slt)
- Load/Store? lw $t2, 32($t3) → Compute address

Stage 4: Memory access



- Used by load and store instructions only
- Other instructions will skip this stage

Stage 5: Writeback



- Write to register file
  - For arithmetic ops, logic, shift, etc, load. What about stores?
- Update PC
  - For branches, jumps

Takeaway

The datapath for a MIPS processor has five stages:

1. Instruction Fetch
2. Instruction Decode
3. Execution (ALU)
4. Memory Access
5. Register Writeback

This five stage datapath is used to execute all MIPS instructions

Next Goal

Specific datapaths MIPS Instructions

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

Instruction Types

Arithmetic

- add, subtract, shift left, shift right, multiply, divide

Memory

- load value from memory to a register
- store value to memory from a register

Control flow

- unconditional jumps
- conditional jumps (branches)
- jump and link (subroutine call)

Many other instructions are possible

- vector add/sub/mul/div, string operations
- manipulate coprocessor
- I/O

MIPS Instruction Types

Arithmetic/Logical

- R-type: result and two source registers, shift amount
- I-type: 16-bit immediate with sign/zero extension

Memory Access

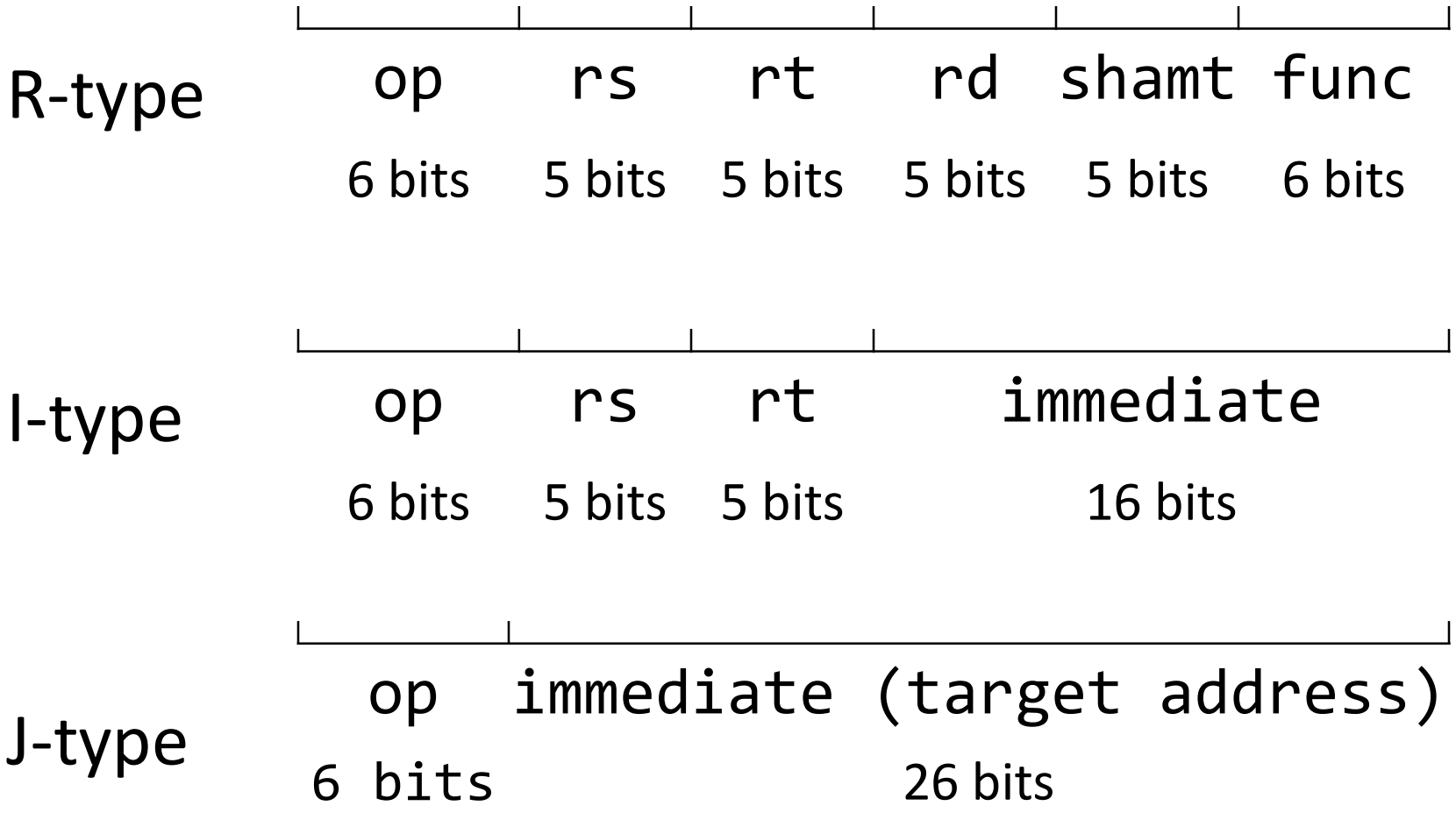
- I-type
- load/store between registers and memory
- word, half-word and byte operations

Control flow

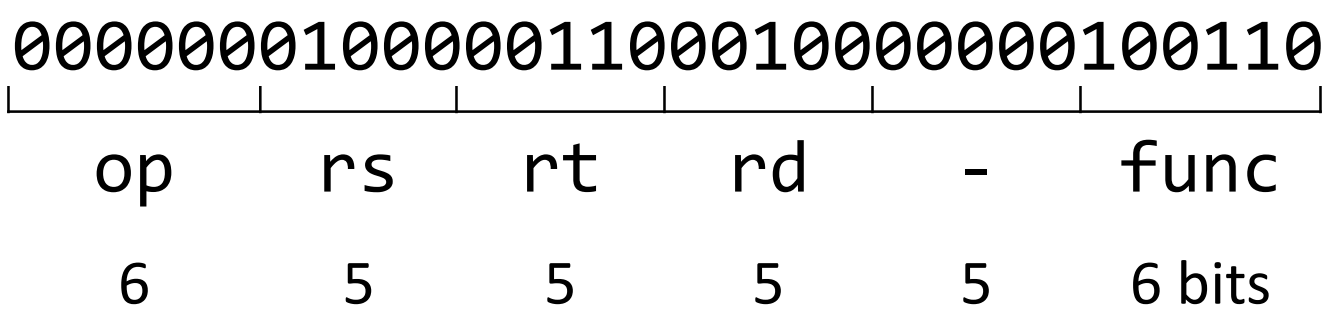
- J-type: fixed offset jumps, jump-and-link
- R-type: register absolute jumps
- I-type: conditional branches: pc-relative addresses

MIPS instruction formats

All MIPS instructions are 32 bits long, has 3 formats

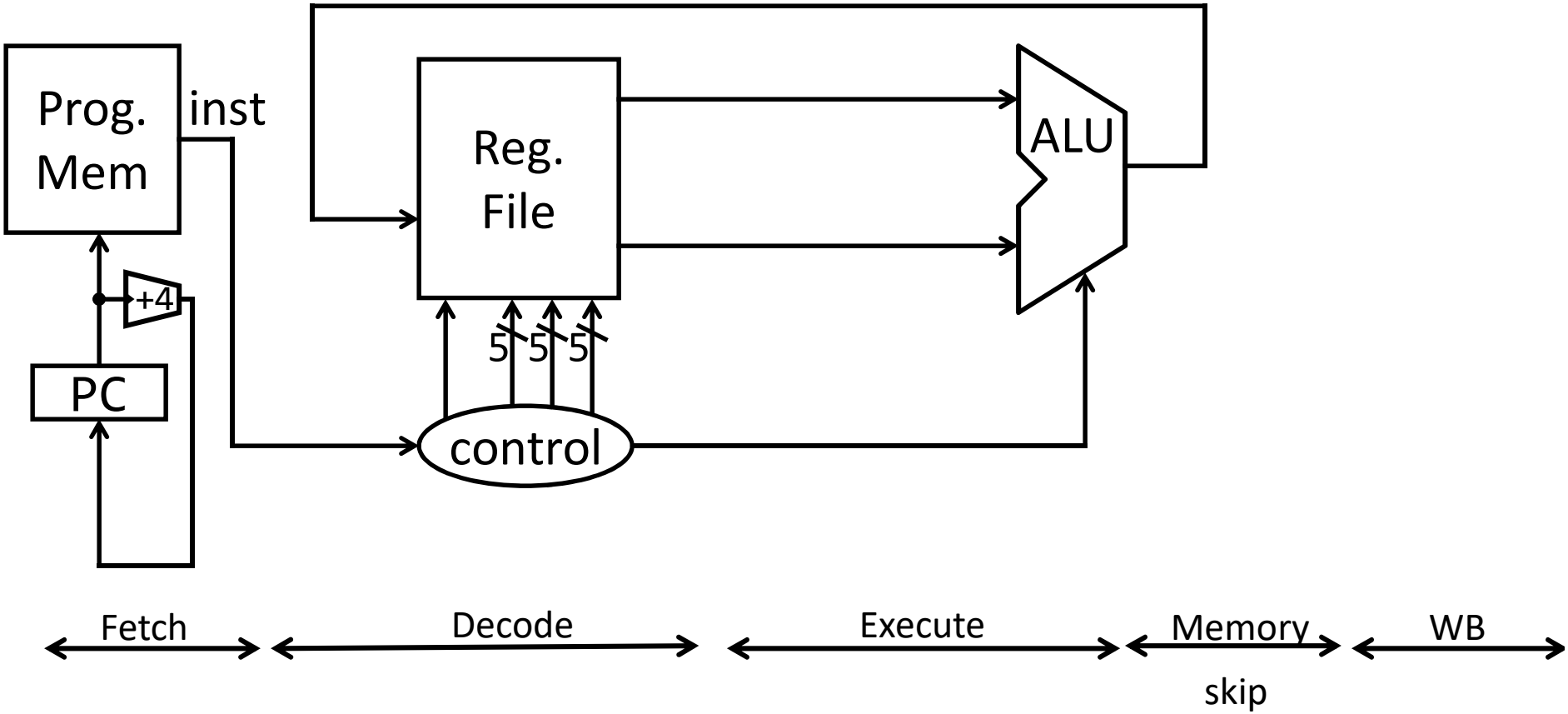


R-Type (1): Arithmetic and Logic

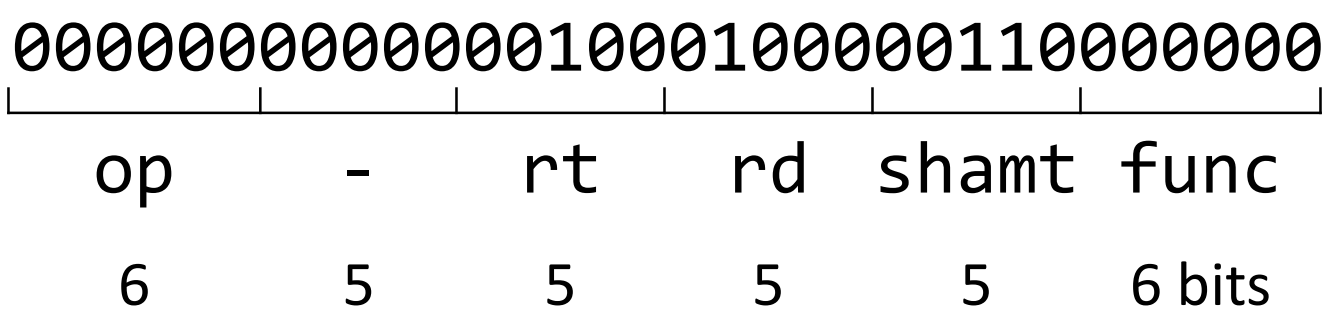


op	func	mnemonic	description
0x0	0x21	ADDU rd, rs, rt	R[rd] = R[rs] + R[rt]
0x0	0x23	SUBU rd, rs, rt	R[rd] = R[rs] – R[rt]
0x0	0x25	OR rd, rs, rt	R[rd] = R[rs] | R[rt]
0x0	0x26	XOR rd, rs, rt	R[rd] = R[rs] ⊕ R[rt]
0x0	0x27	NOR rd, rs rt	R[rd] = ~ (R[rs] | R[rt])

Arithmetic and Logic

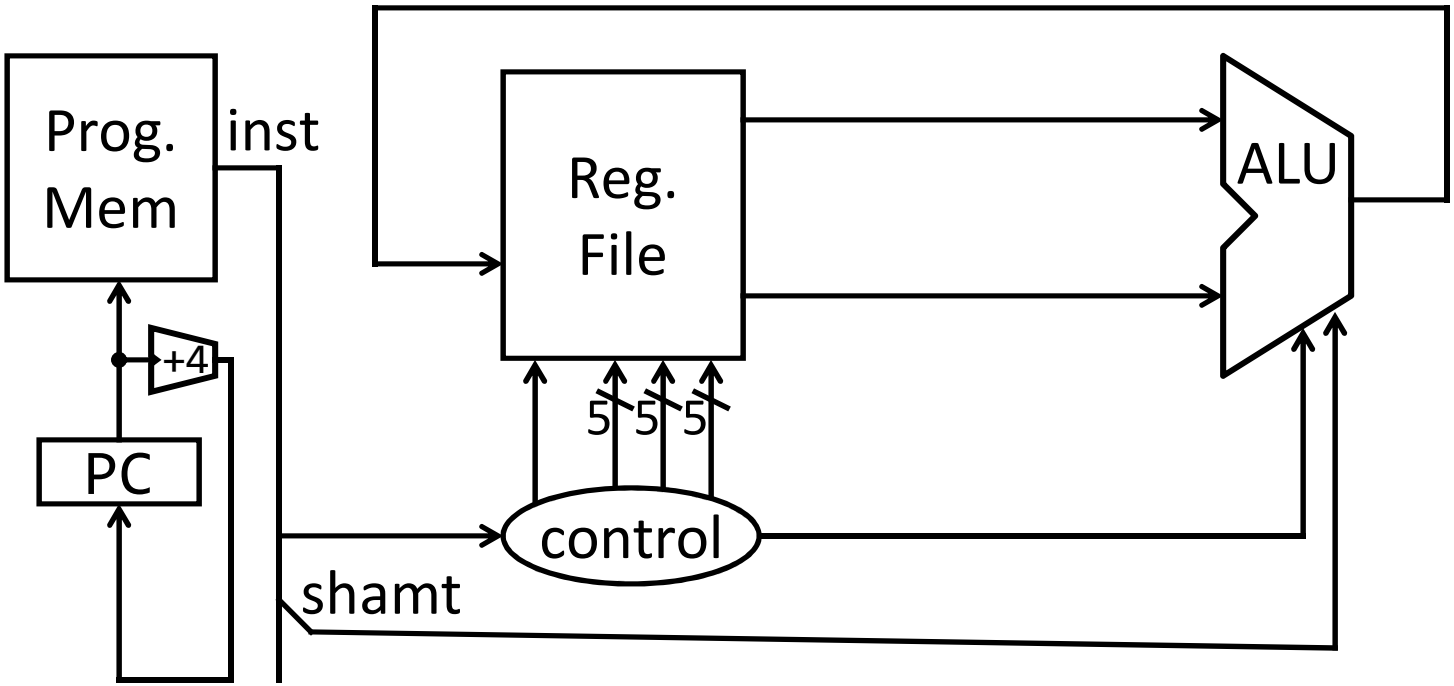


R-Type (2): Shift Instructions

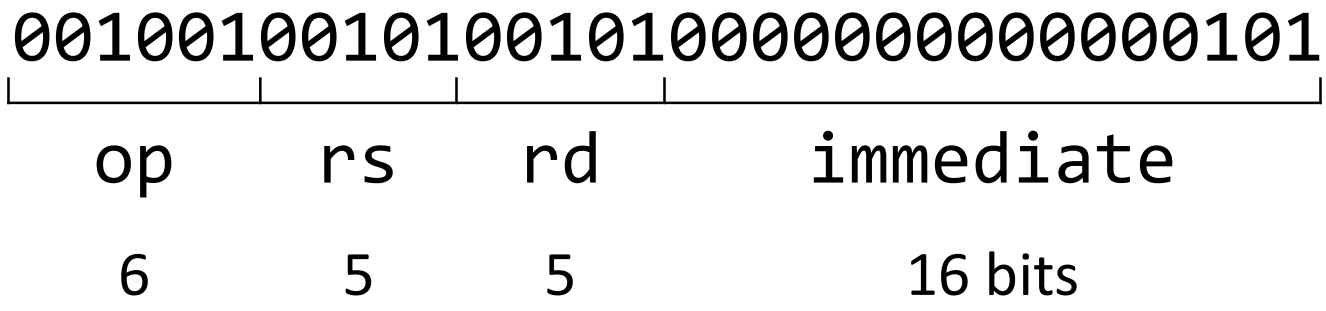


op	func	mnemonic	description
0x0	0x0	SLL rd, rt, shamt	R[rd] = R[rt] << shamt
0x0	0x2	SRL rd, rt, shamt	R[rd] = R[rt] >>> shamt (zero ext.)
0x0	0x3	SRA rd, rt, shamt	R[rd] = R[rt] >> shamt (sign ext.)

Shift

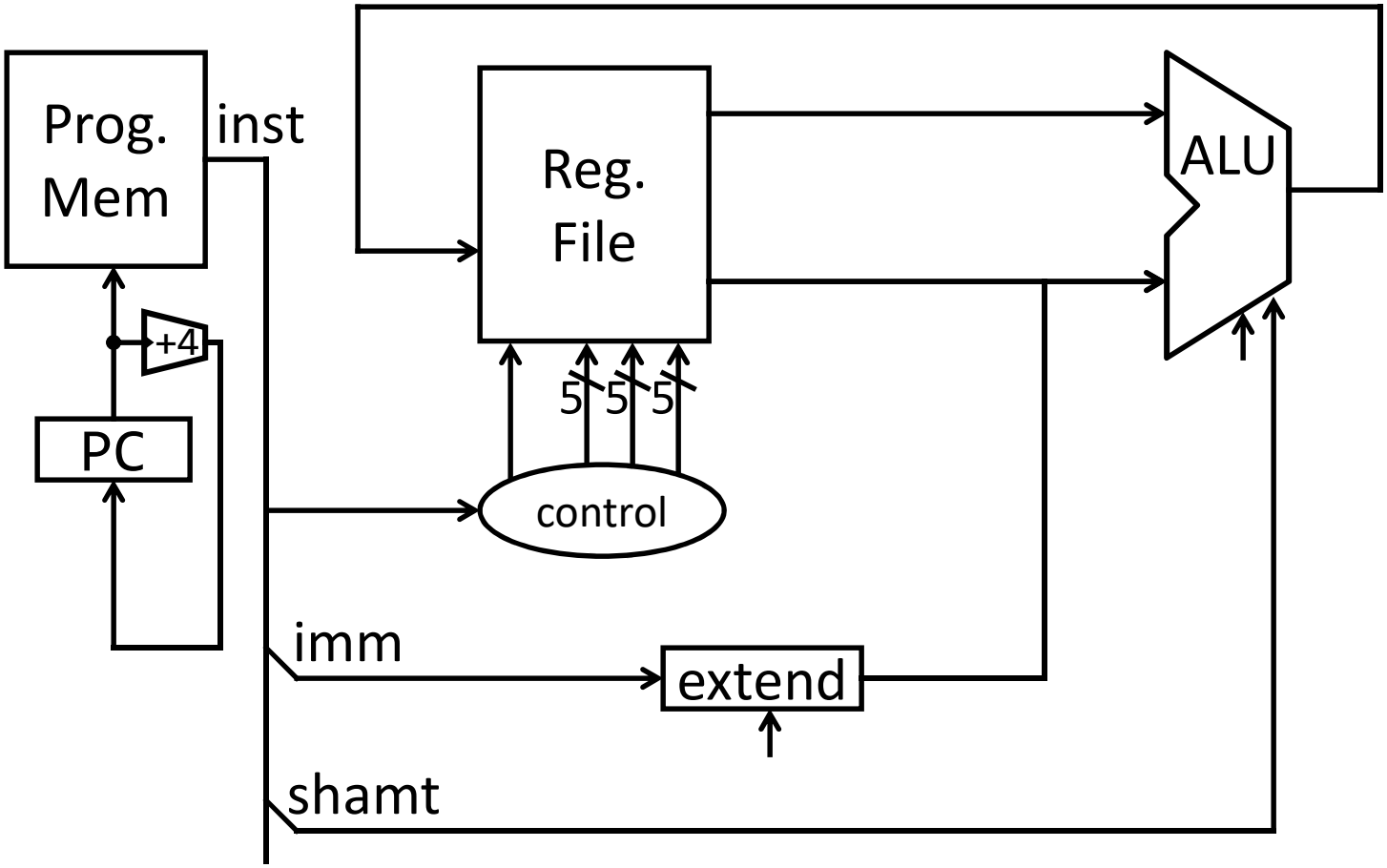


I-Type (1): Arithmetic w/immediates

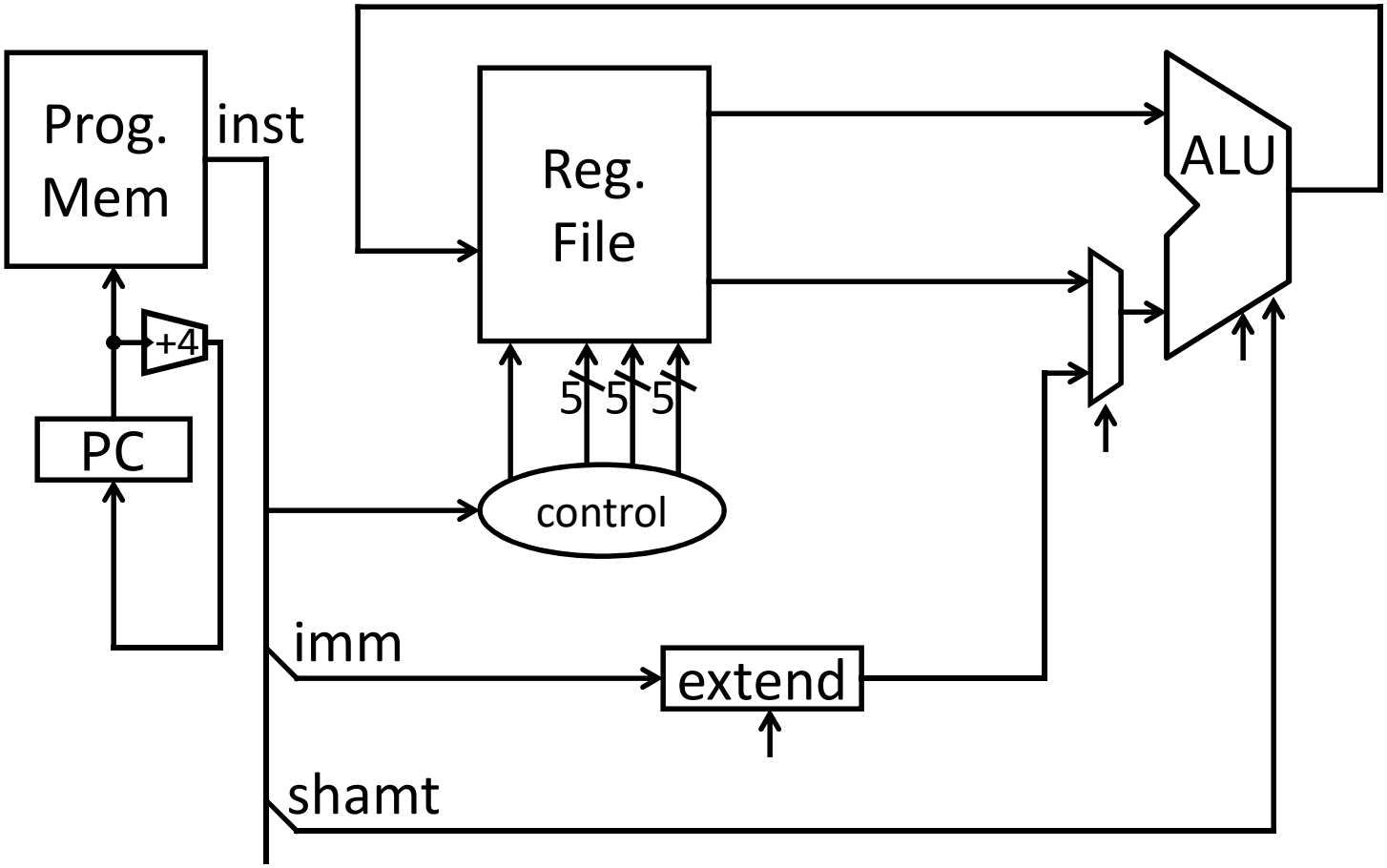


op	mnemonic	description
0x9	ADDIU rd, rs, imm	R[rd] = R[rs] + imm)
0xc	ANDI rd, rs, imm	R[rd] = R[rs] & zero_extend(imm)
0xd	ORI rd, rs, imm	R[rd] = R[rs] | zero_extend(imm)

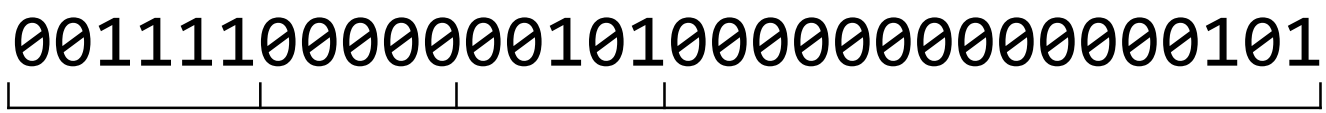
Immediates



Immediates

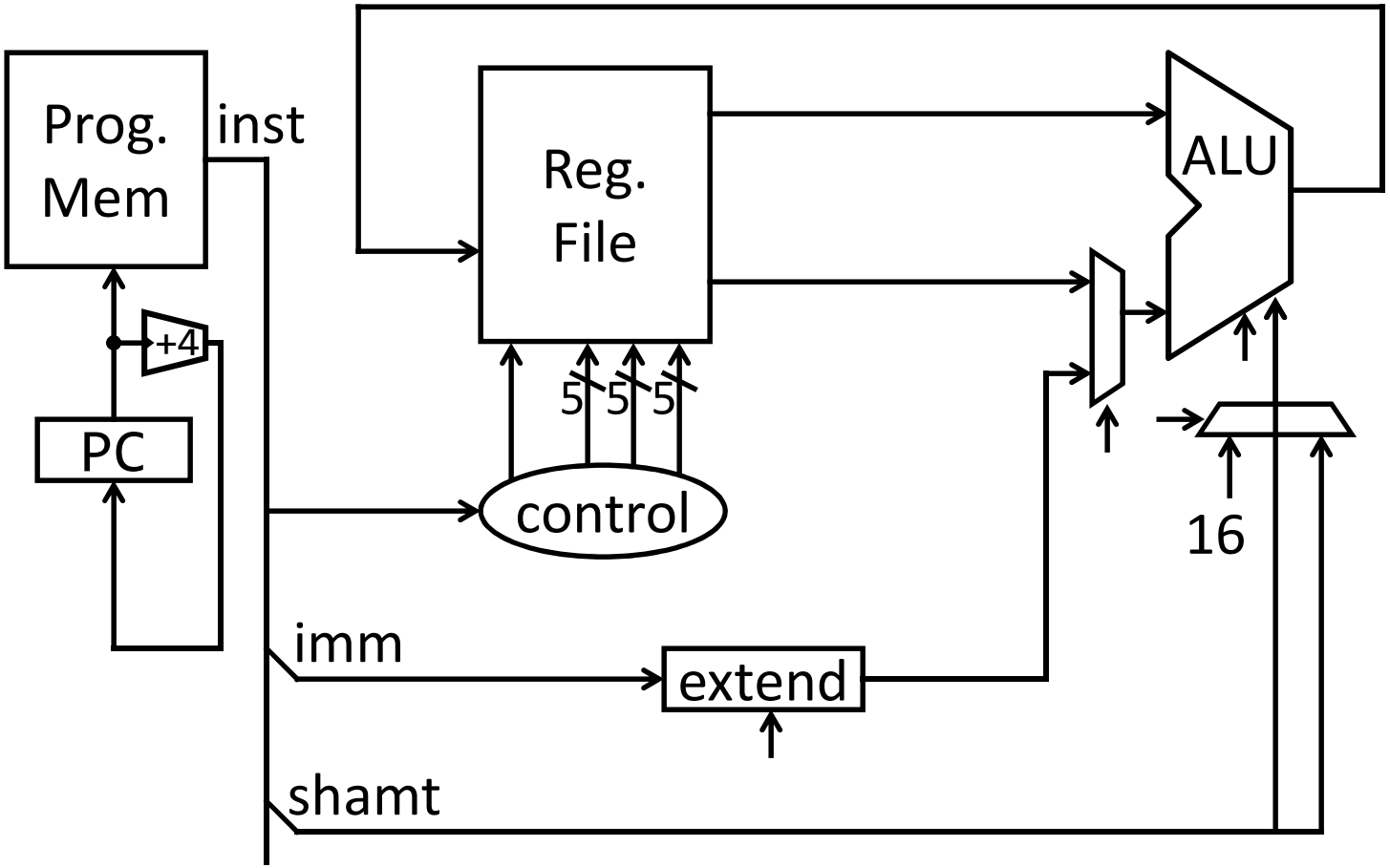


I-Type (2): "Load" Upper Immediate



op	mnemonic	description
0xF	LUI rd, imm	R[rd] = imm << 16

Immediates


Prog.
Mem
inst
Reg.
File
ALU
+4
PC
5
5
5
control
16
imm
extend
shamt

MIPS Instruction Types

Arithmetic/Logical

- R-type: result and two source registers, shift amount
- I-type: 16-bit immediate with sign/zero extension

Memory Access

- I-type
- load/store between registers and memory
- word, half-word and byte operations

Control flow

- J-type: fixed offset jumps, jump-and-link
- R-type: register absolute jumps
- I-type: conditional branches: pc-relative addresses

Memory Layout Options

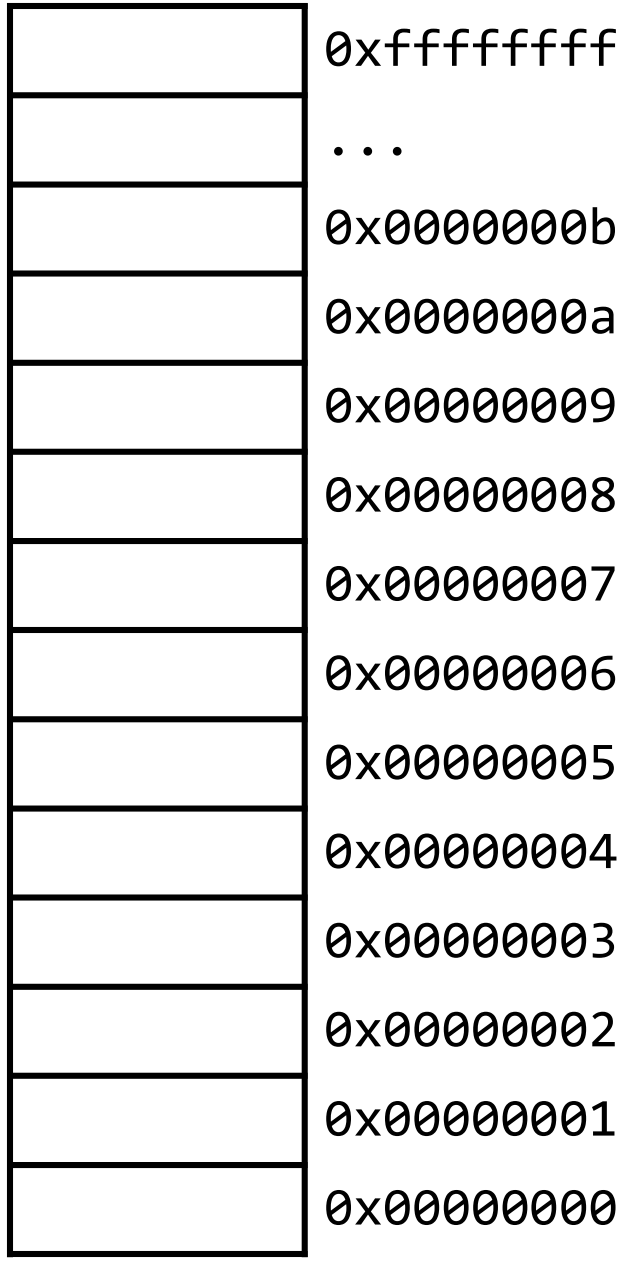
r5 contains 5 (0x00000005)

SB r5, 0(r0)

SB r5, 2(r0)

SW r5, 8(r0)

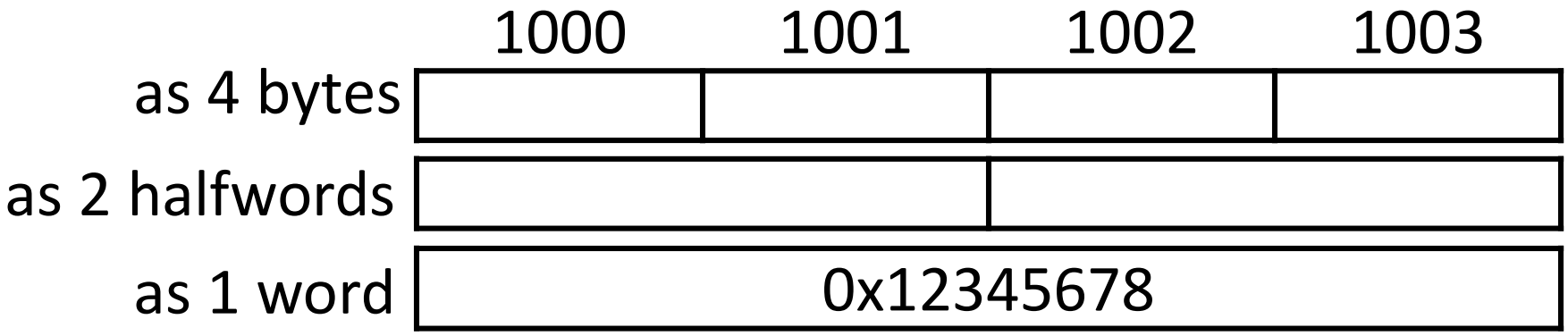
Two ways to store a word in memory.



Little Endian

Endianness: Ordering of bytes within a memory word

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

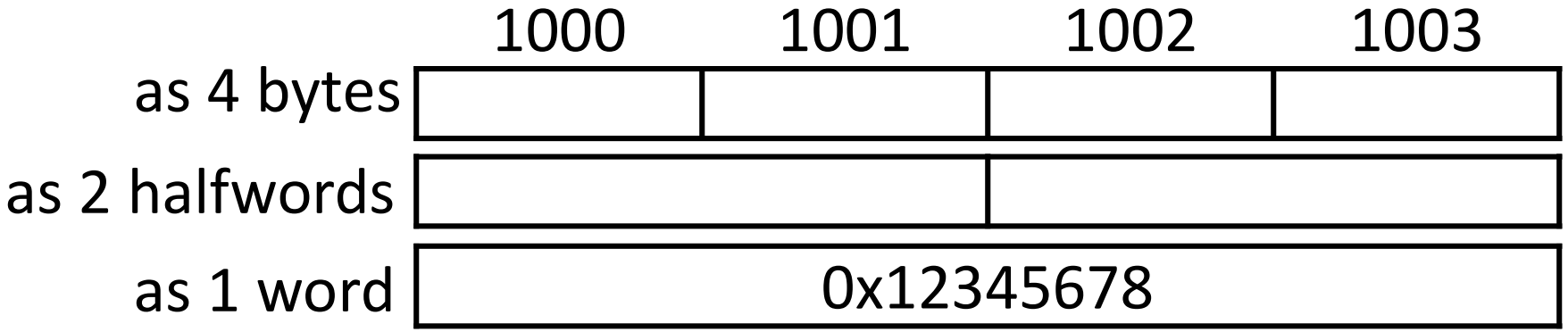


THIS IS WHAT YOUR PROJECTS WILL BE

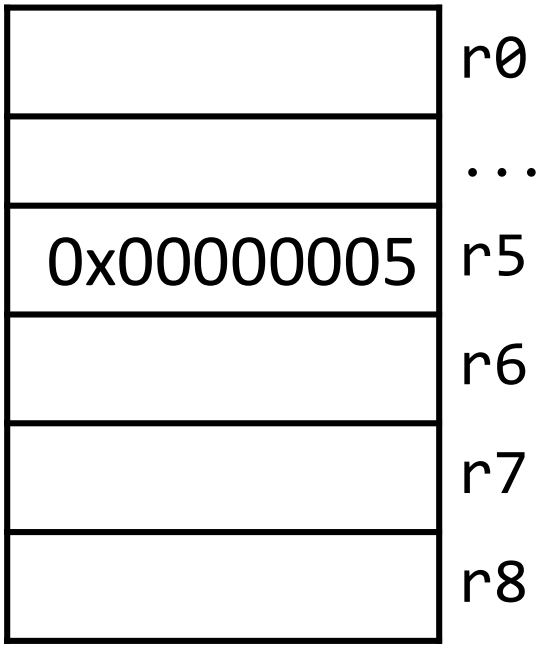
Big Endian

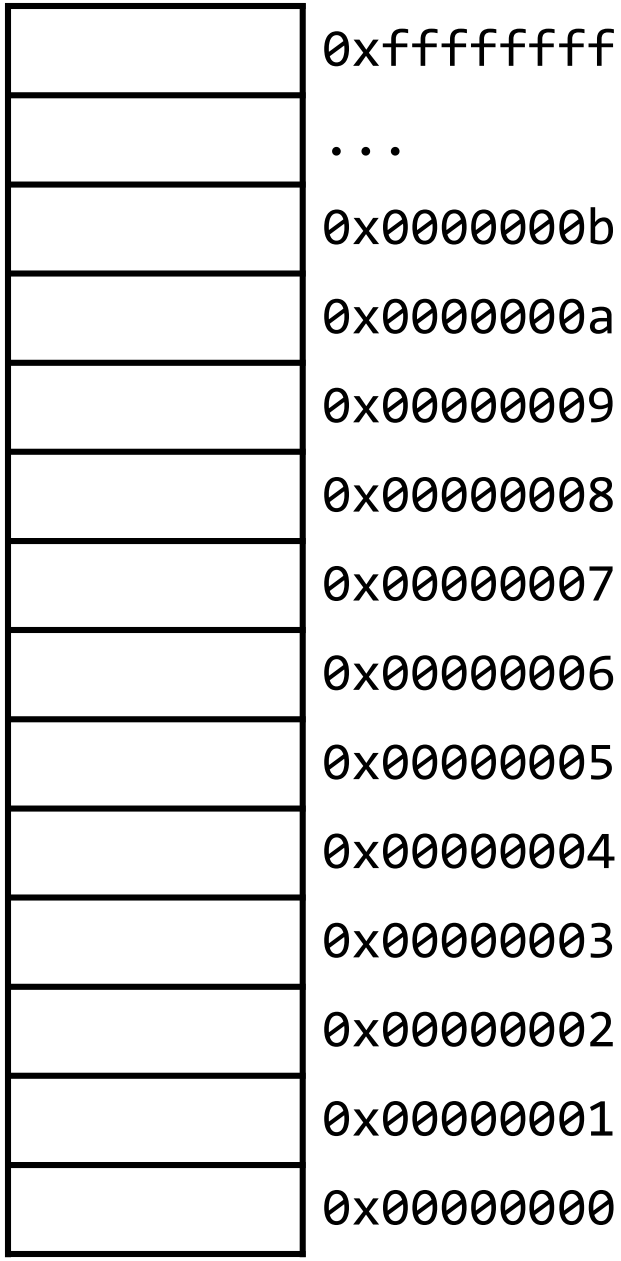
Endianness: Ordering of bytes within a memory word

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



Big Endian Memory Layout





SB r5, 2(r0)

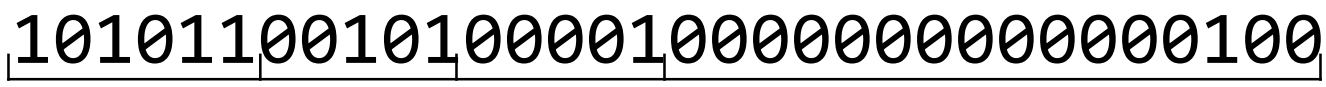
LB r6, 2(r0)

SW r5, 8(r0)

LB r7, 8(r0)

LB r8, 11(r0)

I-Type (3): Memory Instructions

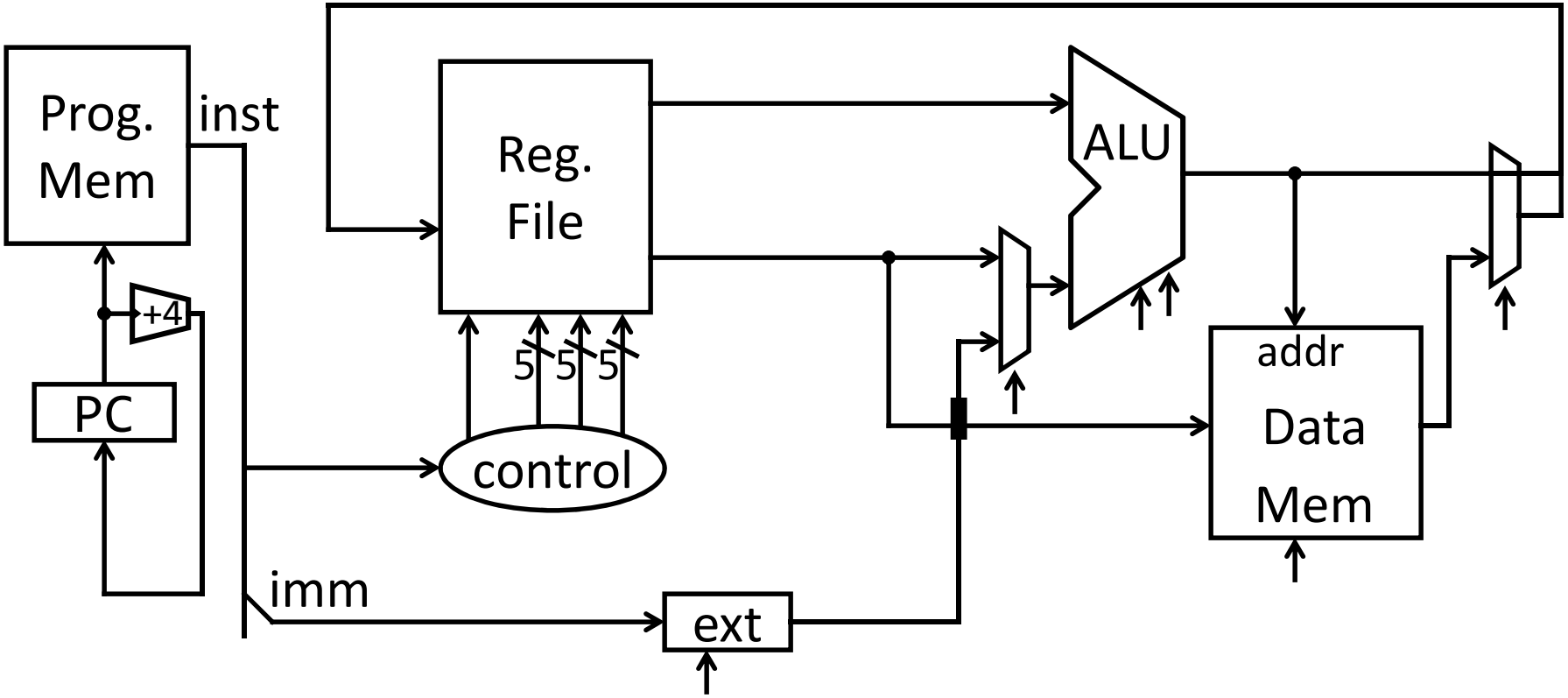


base + offset addressing

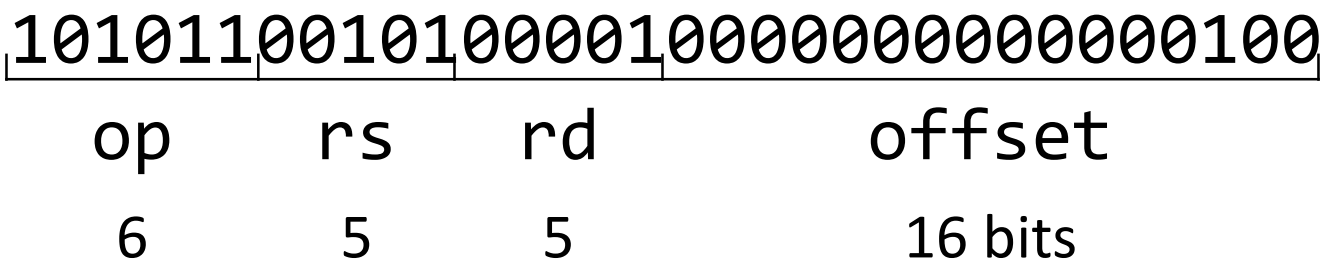
op	mnemonic	description
0x23	LW rd, offset(rs)	R[rd] = Mem[offset+R[rs]]
0x2b	SW rd, offset(rs)	Mem[offset+R[rs]] = R[rd]

signed offsets

Memory Operations



More Memory Instructions



op	mnemonic	description
0x20	LB rd, offset(rs)	R[rd] = sign_ext(Mem[offset+R[rs]])
0x24	LBU rd, offset(rs)	R[rd] = zero_ext(Mem[offset+R[rs]])
0x21	LH rd, offset(rs)	R[rd] = sign_ext(Mem[offset+R[rs]])
0x25	LHU rd, offset(rs)	R[rd] = zero_ext(Mem[offset+R[rs]])
0x23	LW rd, offset(rs)	R[rd] = Mem[offset+R[rs]]
0x28	SB rd, offset(rs)	Mem[offset+R[rs]] = R[rd]
0x29	SH rd, offset(rs)	Mem[offset+R[rs]] = R[rd]
0x2b	SW rd, offset(rs)	Mem[offset+R[rs]] = R[rd]

MIPS Instruction Types

Arithmetic/Logical

- R-type: result and two source registers, shift amount
- I-type: 16-bit immediate with sign/zero extension

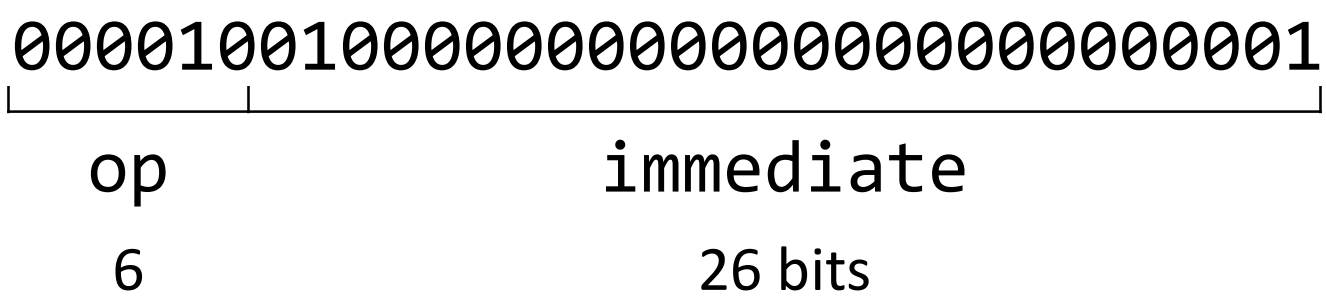
Memory Access

- I-type
- load/store between registers and memory
- word, half-word and byte operations

Control flow

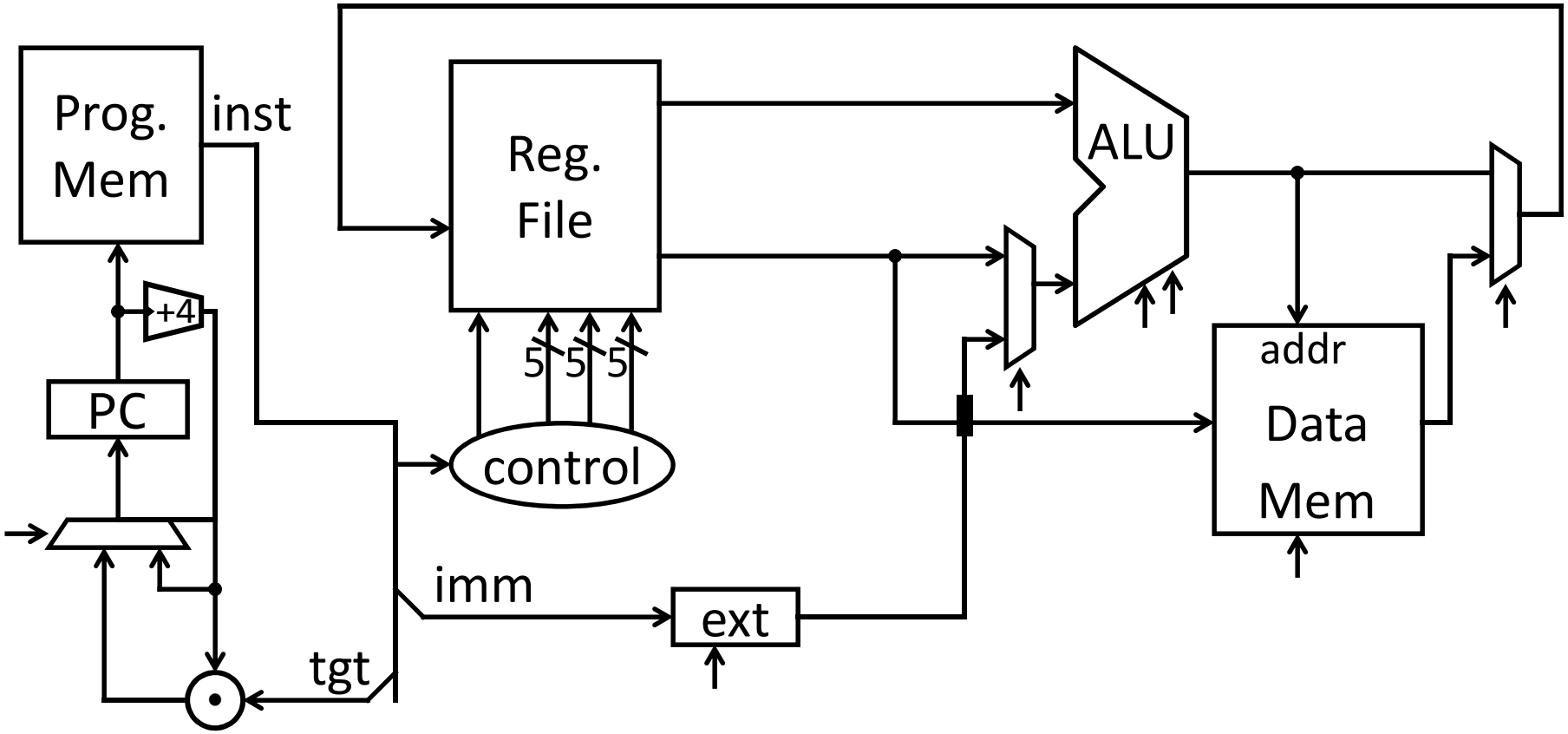
- J-type: fixed offset jumps, jump-and-link
- R-type: register absolute jumps
- I-type: conditional branches: pc-relative addresses

J-Type (1): Absolute Jump

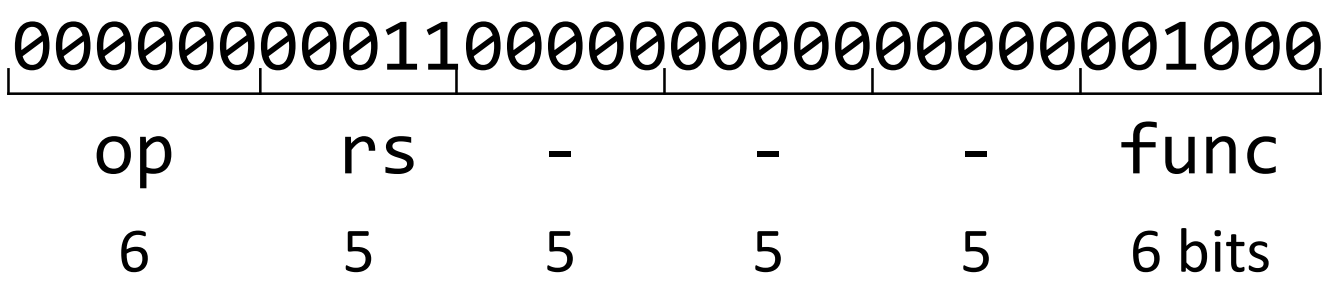


op	Mnemonic	Description	⧆ = *concatenate*
0x2	J target	$PC = (PC+4)_{31..28}$ • target • 00	

Absolute Jump

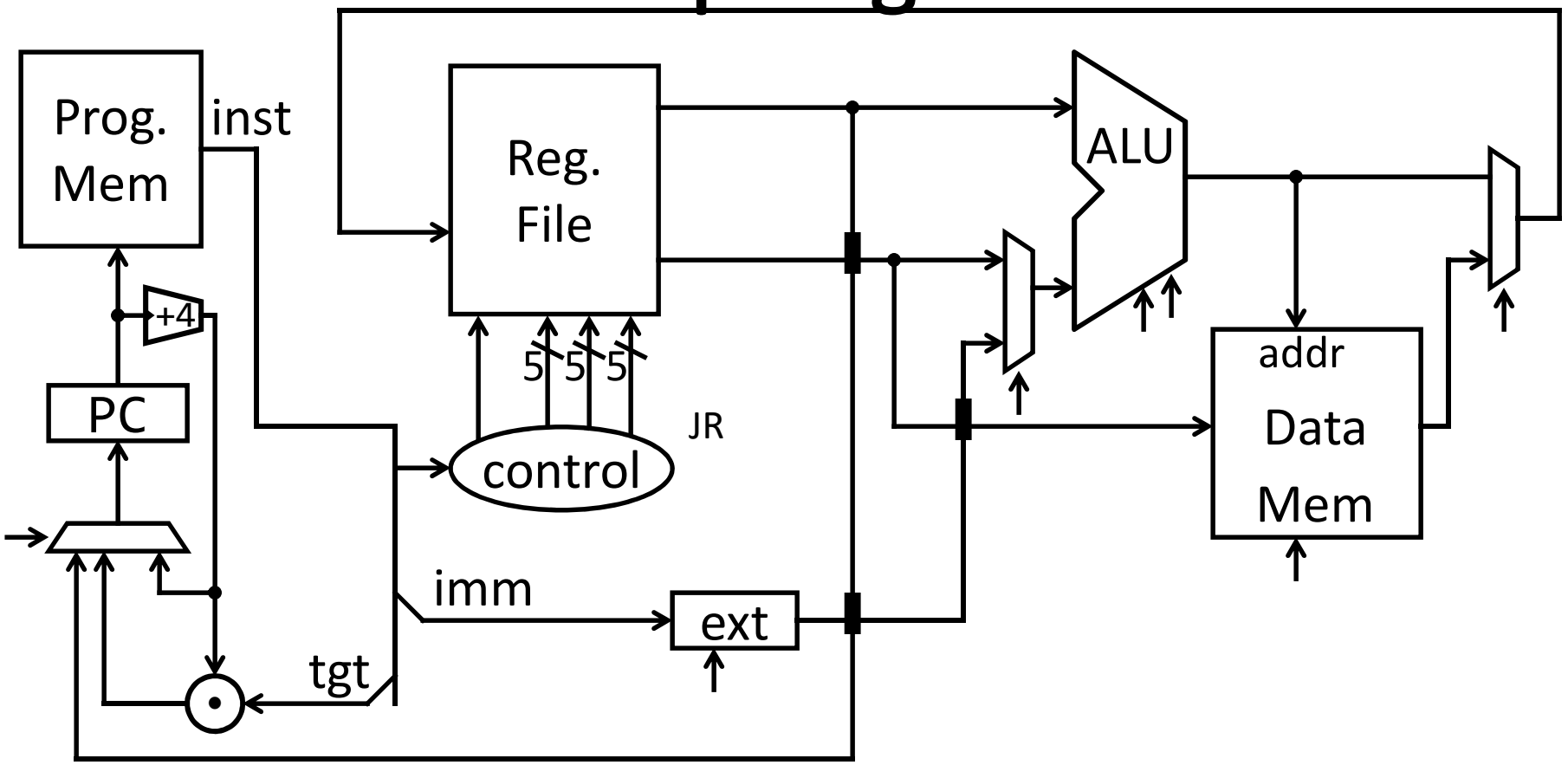


R-Type (3): Jump Register



op	func	mnemonic	description
0x0	0x08	JR rs	PC = R[rs]

Jump Register



Moving Beyond Jumps

Can use Jump or Jump Register instruction to jump to 0xabcd1234

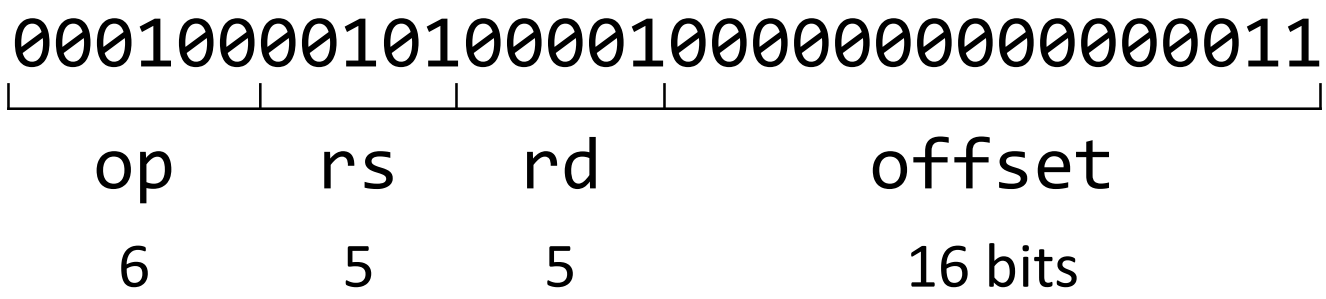
What about a jump based on a condition?

assume 0 <= r3 <= 1

if (r3 == 0) jump to 0xdecafe00

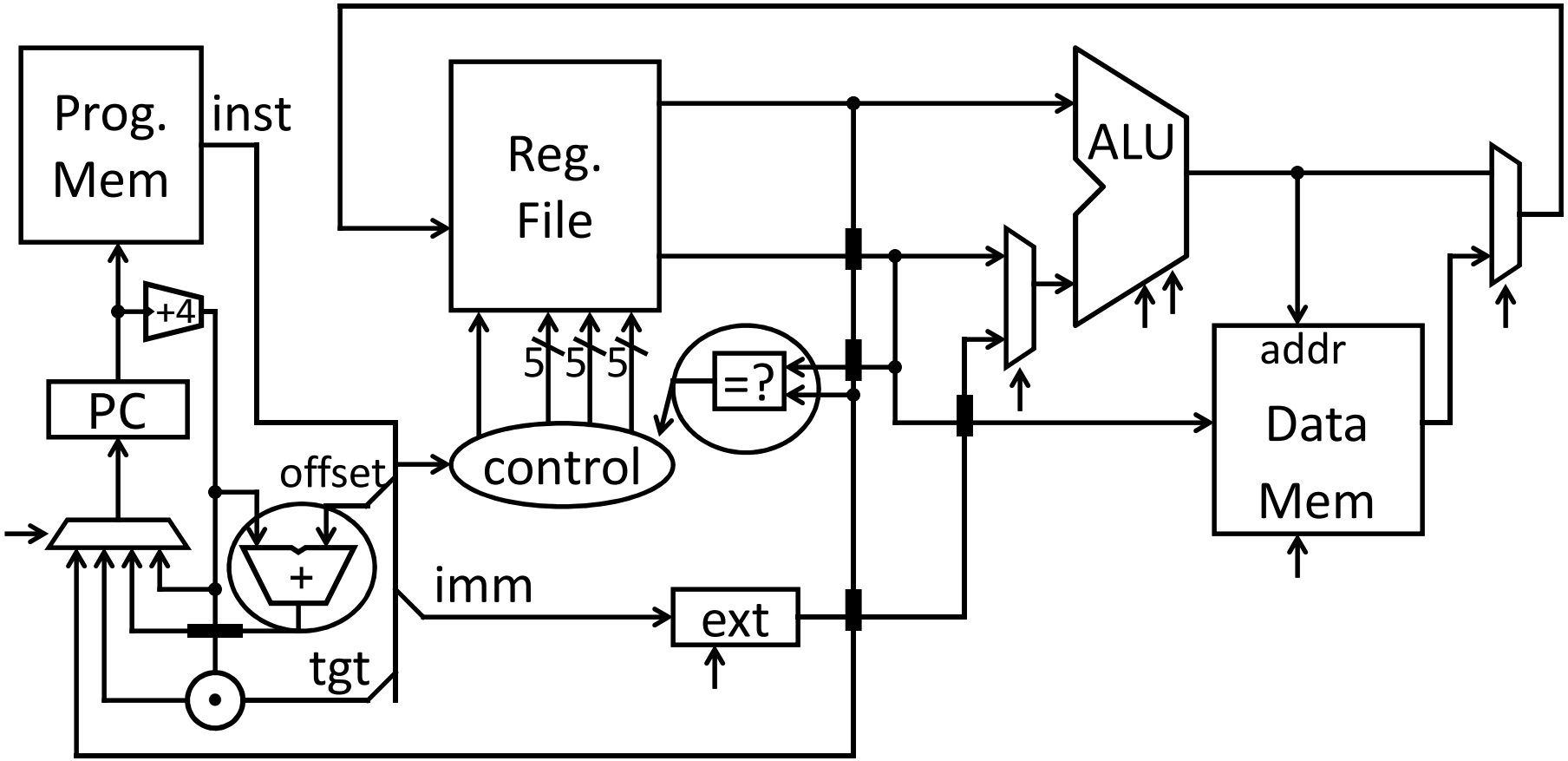
else jump to 0xabcd1234

I-Type (4): Branches

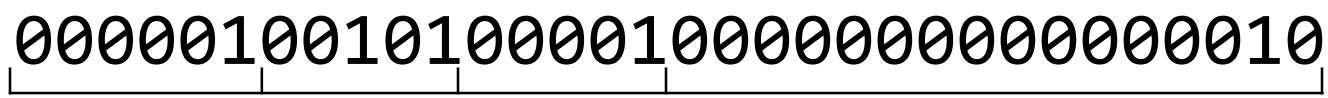


op	mnemonic	description
0x4	BEQ rs, rd, offset	if R[rs] == R[rd] then PC = PC+4 + (offset<<2)
0x5	BNE rs, rd, offset	if R[rs] != R[rd] then PC = PC+4 + (offset<<2)

Control Flow: Branches



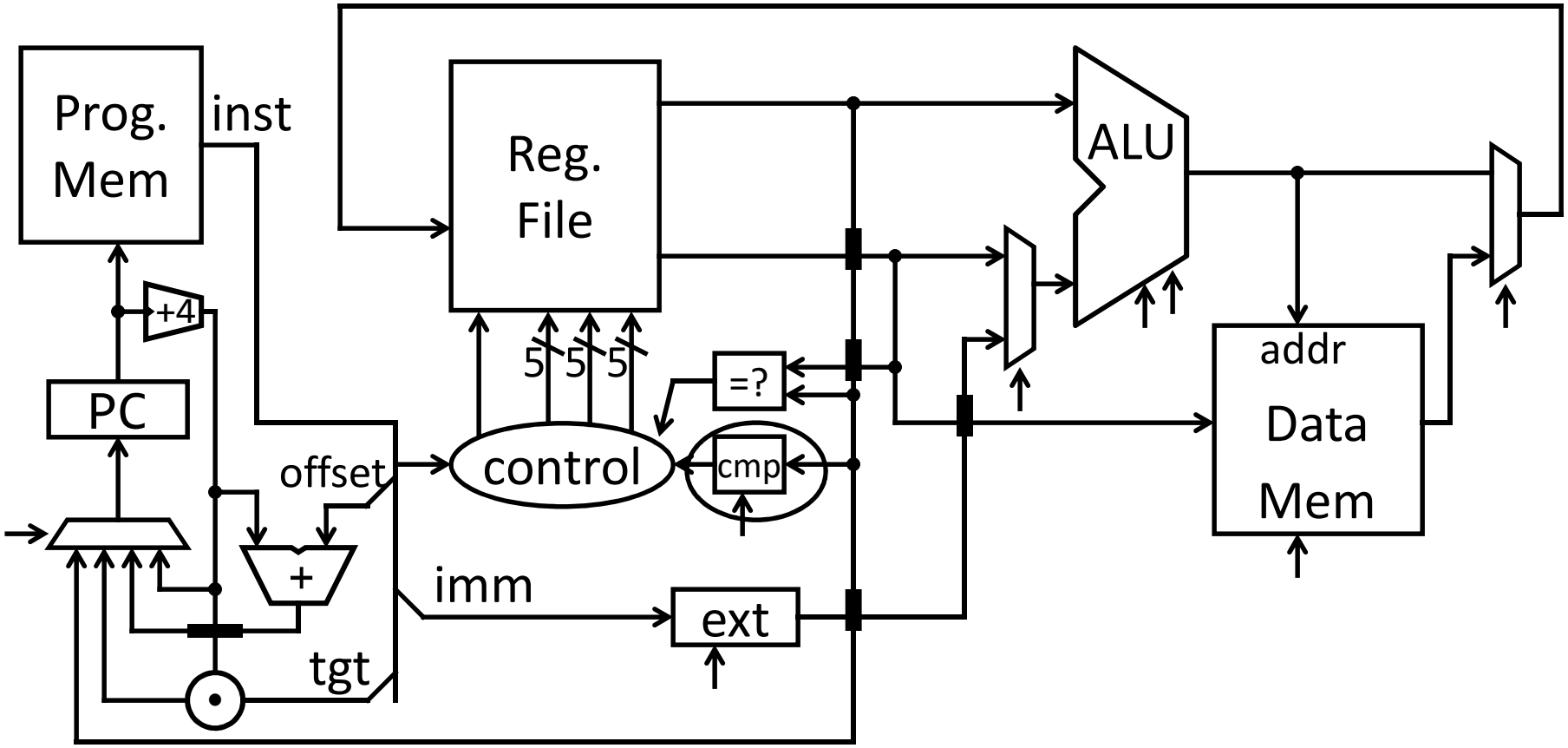
I-Type (5): Conditional Jumps



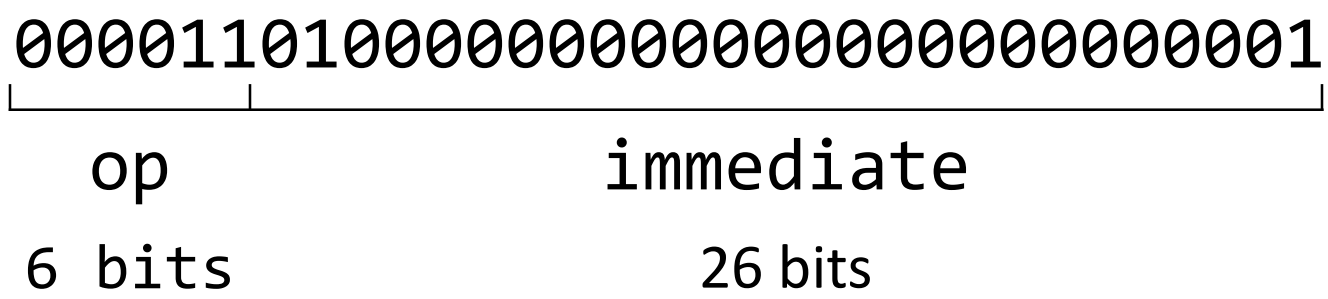
signed

op	subop	mnemonic	description
0x1	0x0	BLTZ rs, offset	if R[rs] < 0 then PC = PC+4+ (offset<<2)
0x1	0x1	BGEZ rs, offset	if R[rs] ≥ 0 then PC = PC+4+ (offset<<2)
0x6	0x0	BLEZ rs, offset	if R[rs] ≤ 0 then PC = PC+4+ (offset<<2)
0x7	0x0	BGTZ rs, offset	if R[rs] > 0 then PC = PC+4+ (offset<<2)

Control Flow: More Branches



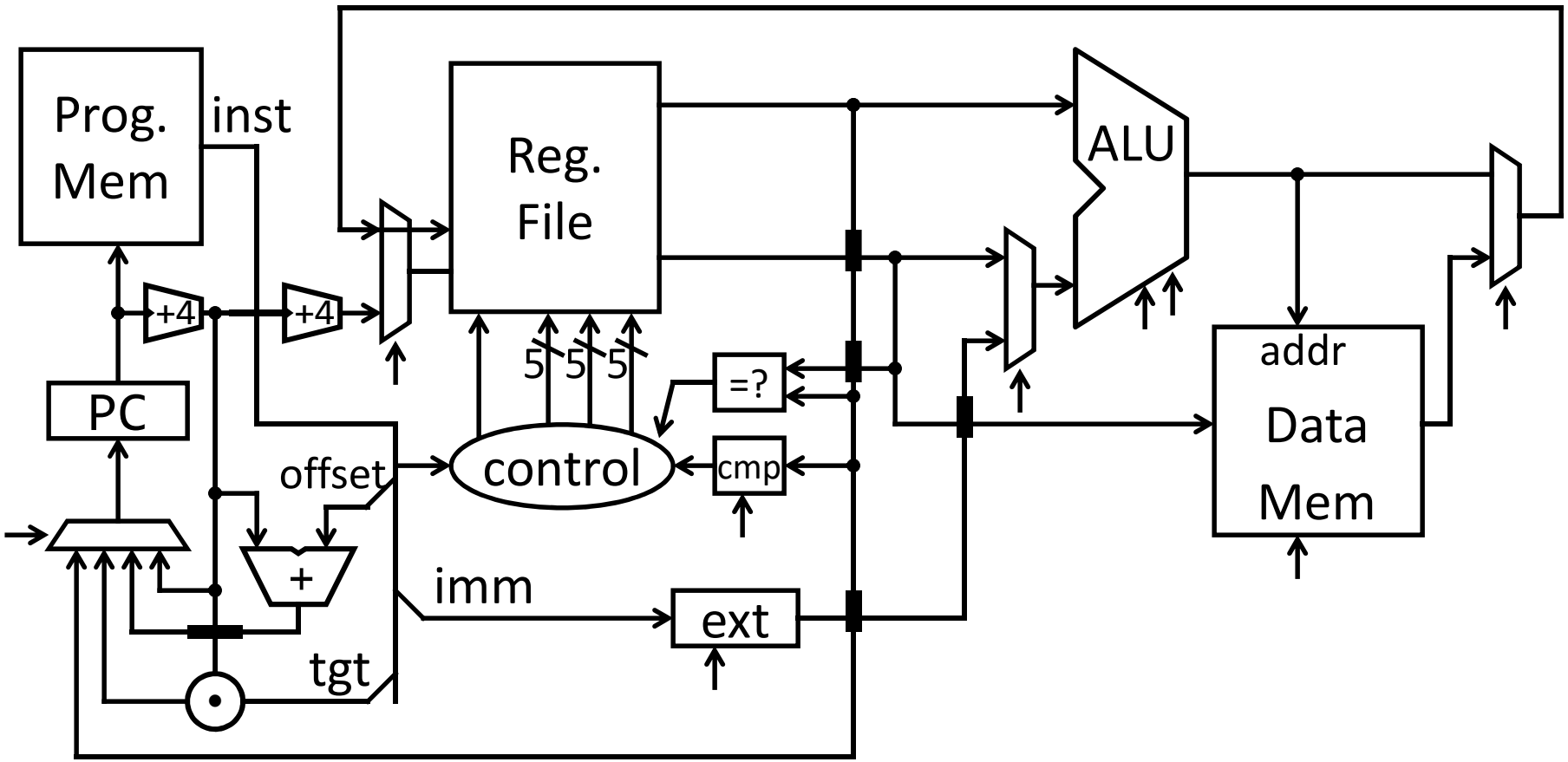
J-Type (2): Jump and Link



op	mnemonic	description
0x3	JAL target	r31 = PC+8 (+8 due to branch delay slot) PC = $(PC+4)_{31..28}$ • target • 00

Discuss later

Jump and Link



MIPS Instruction Types

✓ Arithmetic/Logical
- R-type: result and two source registers, shift amount
- I-type: 16-bit immediate with sign/zero extension

✓ Memory Access
- I-type
- load/store between registers and memory
- word, half-word and byte operations

✓ Control flow
- J-type: fixed offset jumps, jump-and-link
- R-type: register absolute jumps
- I-type: conditional branches: pc-relative addresses

Many other instructions possible:
- vector add/sub/mul/div, string operations
- manipulate coprocessor
- I/O

Summary

We have all that it takes to build a processor!

- Arithmetic Logic Unit (ALU)
- Register File
- Memory

MIPS processor and ISA is an example of a Reduced Instruction Set Computers (RISC).

Simplicity is key, thus enabling us to build it!

We now know the data path for the MIPS ISA:

- register, memory and control instructions