The MIPS Processor

Anne Bracy

CS 3410

Computer Science

Cornell University

[K. Bala, A. Bracy, E. Sirer, and H. Weatherspoon]

Goal for this lecture

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
- MIPS Instructions & how they are executed

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

```
00100000000000100000000000001010
00100000000000010000000000000000
00000000001000100001100000101010
```

Machine Language

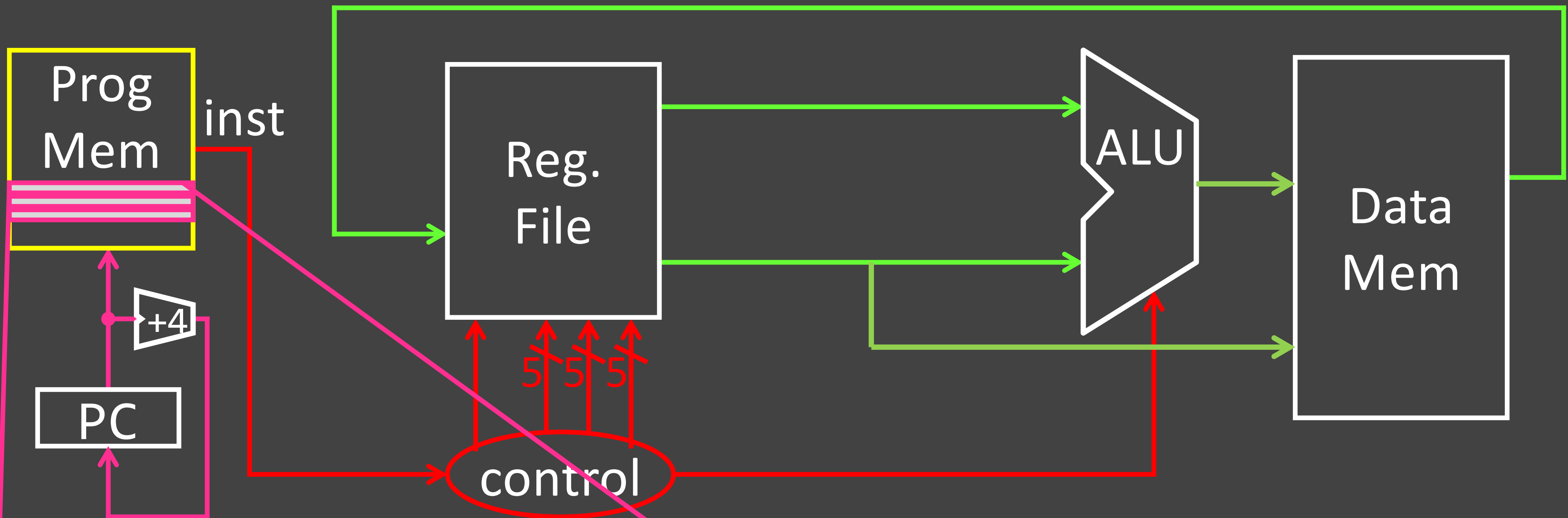
- Binary-encoded assembly
- Labels become addresses
- **The language of the CPU**

Instruction Set Architecture

ALU, Control, Register File, …

Machine Implementation (Microarchitecture)

Instruction Processing



Instructions:

stored in memory, encoded in binary

00100000000000100000000000001010
00100000000000010000000000000000
00000000001000100001100000101010

A basic processor

- fetches
- decodes
- executes

one instruction at a time

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

R-Type (1): Arithmetic and Logic

```
00000001000001100010000000100110
```

op	rs	rt	rd	-	func
000000	01000	00110	00100	00000	100110
6	5	5	5	5	6 bits

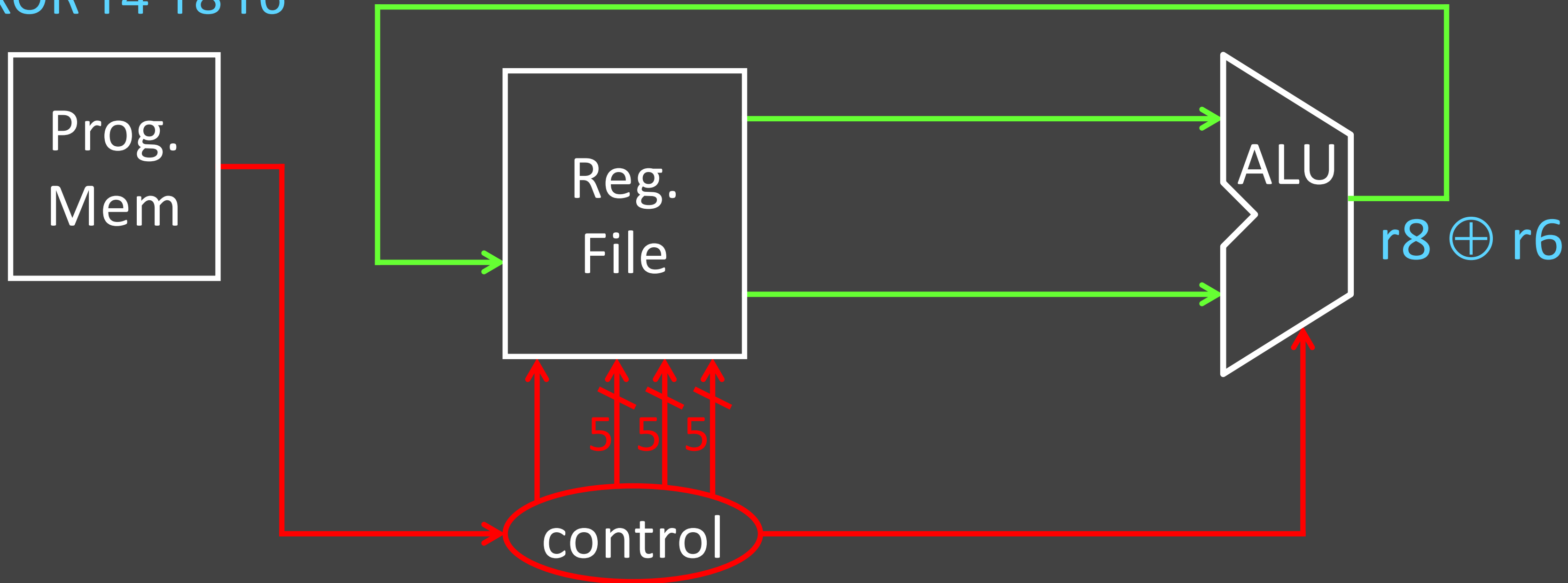
op	func	mnemonic	description
0x0	0x21	ADDU rd, rs, rt	R[rd] = R[rs] + R[rt]
0x0	0x23	SUBU rd, rs, rt	R[rd] = R[rs] – R[rt]
0x0	0x25	OR rd, rs, rt	R[rd] = R[rs] | R[rt]
0x0	0x26	XOR rd, rs, rt	R[rd] = R[rs] ⊕ R[rt]
0x0	0x27	NOR rd, rs rt	R[rd] = ~ (R[rs] | R[rt])

example: r4 = r8 ⊕ r6 # XOR r4, r8, r6

rd, rs, rt

Arithmetic and Logic

XOR r4 r8 r6



Example: $r4 = r8 \oplus r6$ # XOR r4, r8, r6

R-Type (2): Shift Instructions

```
00000000000001000100000110000000
```

op	-	rt	rd	shamt	func
6	5	5	5	5	6 bits

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.)

example: r8 = r4 * 64 # SLL r8, r4, 6

r8 = r4 << 6

Shift

SLL r8 r4 6

Prog. Mem

Reg. File

ALU

r4 << 6

5 5 5

control

Example: r8 = r4 * 64 # SLL r8, r4, 6

r8 = r4 << 6

I-Type (1): Arithmetic w/immediates

```
001001 00101 00101 0000000000000101
op     rs    rd    immediate
6      5     5     16 bits
```

op	mnemonic	description
0x9	ADDIU rd, rs, imm	R[rd] = R[rs] + sign_extend(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)

example: r5 = r5 + 5 # ADDIU r5, r5, 5

r5 += 5

What if immediate is negative?

r5 += -1 r5 += 65535

Unsigned means no overflow detection.
The immediate *can* be negative!

Arithmetic w/immediates

ADDIU r5 r5 5

Prog.
Mem

Reg.
File

ALU

r5 + 5

5 5 5

control

imm

16

extend

32

shamt

Example: r5 = r5 + 5 # ADDIU r5, r5, 5

iClicker Question

To compile the code `y = x + 1`, assuming y is stored in R1 and x is stored in R2, you can use the `ADDI` instruction. What is the largest number for which we can continue to use `ADDI`?

(a) 16

(b) 2^{15-1} = 32,767

(c) 2^{16-1} = 65,535

(d) 2^{31-1} = ~2.1 billion

(e) 2^{32-1} = ~4.3 billion

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

```
00111100000001010000000000000101
```

op	-	rd	immediate
6	5	5	16 bits

WORST NAME EVER!

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

example: r5 = 0x50000 # LUI r5, 5

Example:
```
LUI r5, 0xdead
ORI r5, r5 0xbeef
```

What does r5 = ?

Load Upper Immediate

LUI r5 5

Prog. Mem

Reg. File

ALU

0x50000

control

5 5 5

16

imm

16

extend

32

shamt

Example: r5 = 0x50000 # LUI r5, 5

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

I-Type (3): Memory Instructions

```
101011 00101 00001 0000000000000100
  op     rs    rd        offset
  6      5     5        16 bits
```

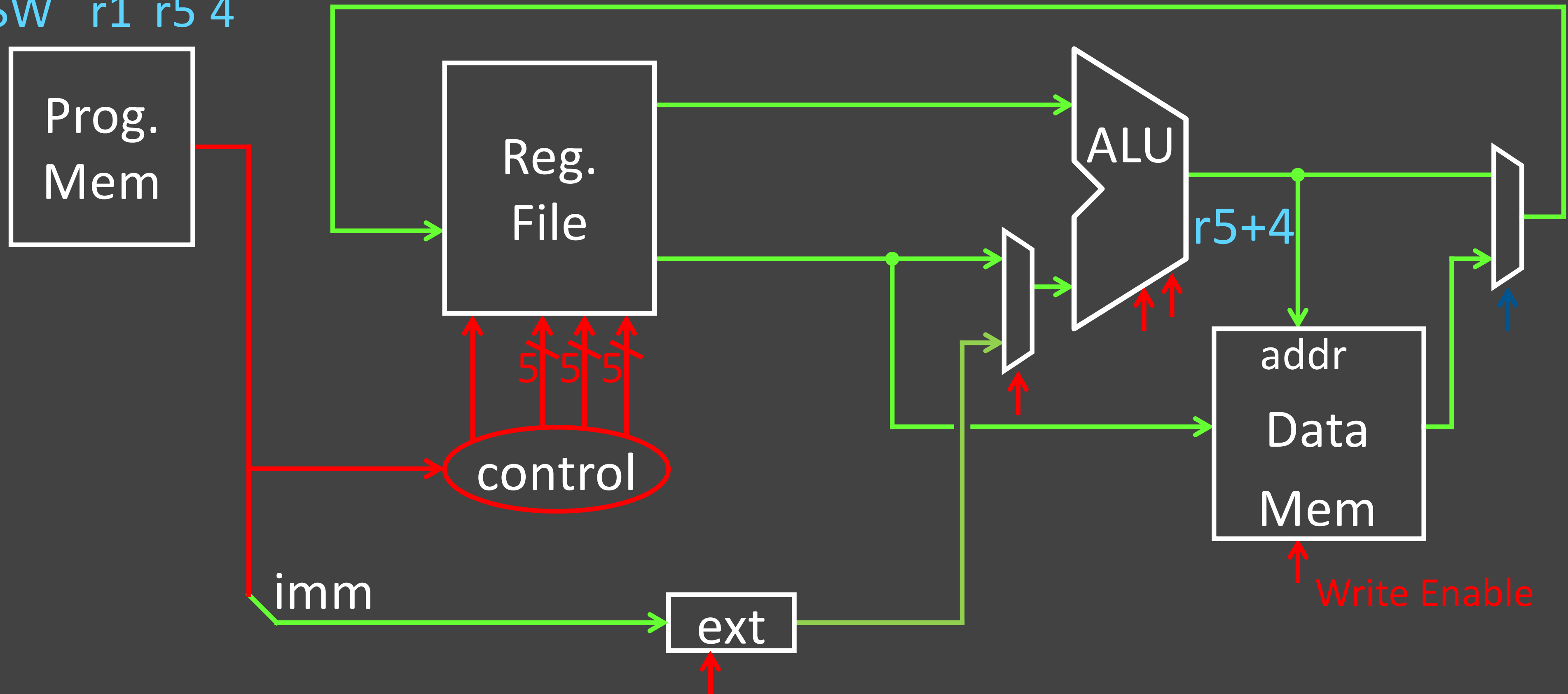
base + offset addressing

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

Example: = Mem[4+r5] = r1 # SW r1, 4(r5)

Memory Operations



Example: = Mem[4+r5] = r1 # SW r1, 4(r5)

More Memory Instructions

```
101011 00101 00001 0000000000000100
op     rs    rd    offset
6      5     5     16 bits
```

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]

Memory Layout Options

r5 contains 5 (0x00000005)

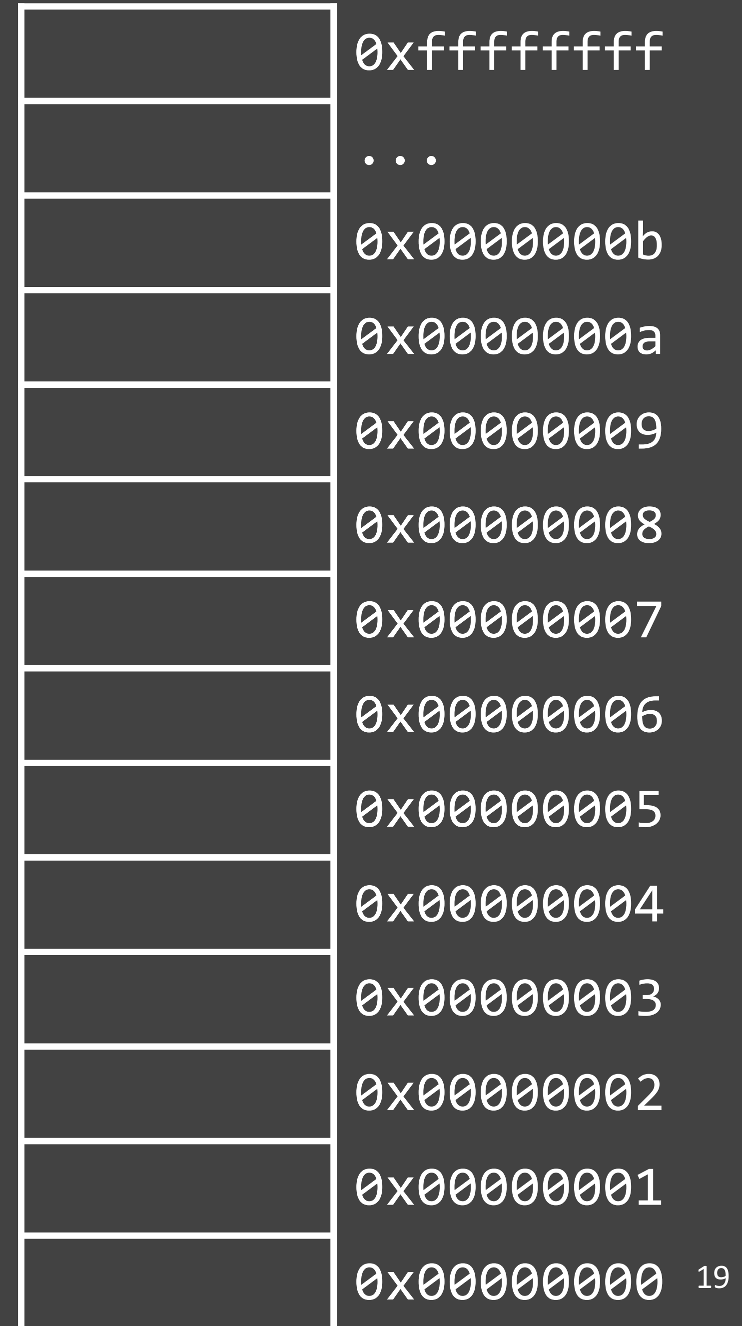
SB r5, 0(r0)

SB r5, 2(r0)

SW r5, 8(r0)

Two ways to store a word in memory.

Endianness: ordering of bytes within a memory word



WHAT WE USE IN 3410

Little Endian

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

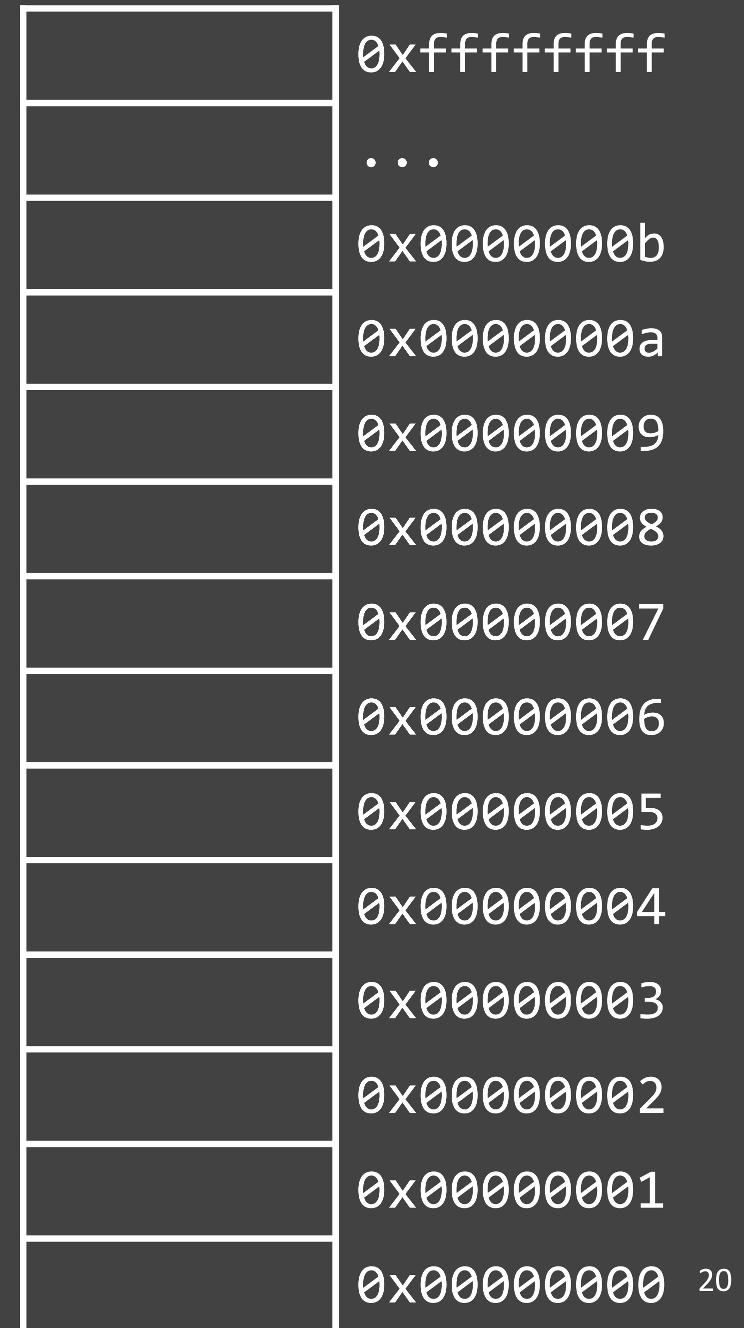
Example:

r5 contains 5 (0x00000005)

```
SW r5, 8(r0)
```

Clicker Question: After executing the store, which byte address contains the byte 0x05?

a) 0x00000008
b) 0x00000008
c) 0x00000008
d) 0x00000008
e) I don't know



Big Endian

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

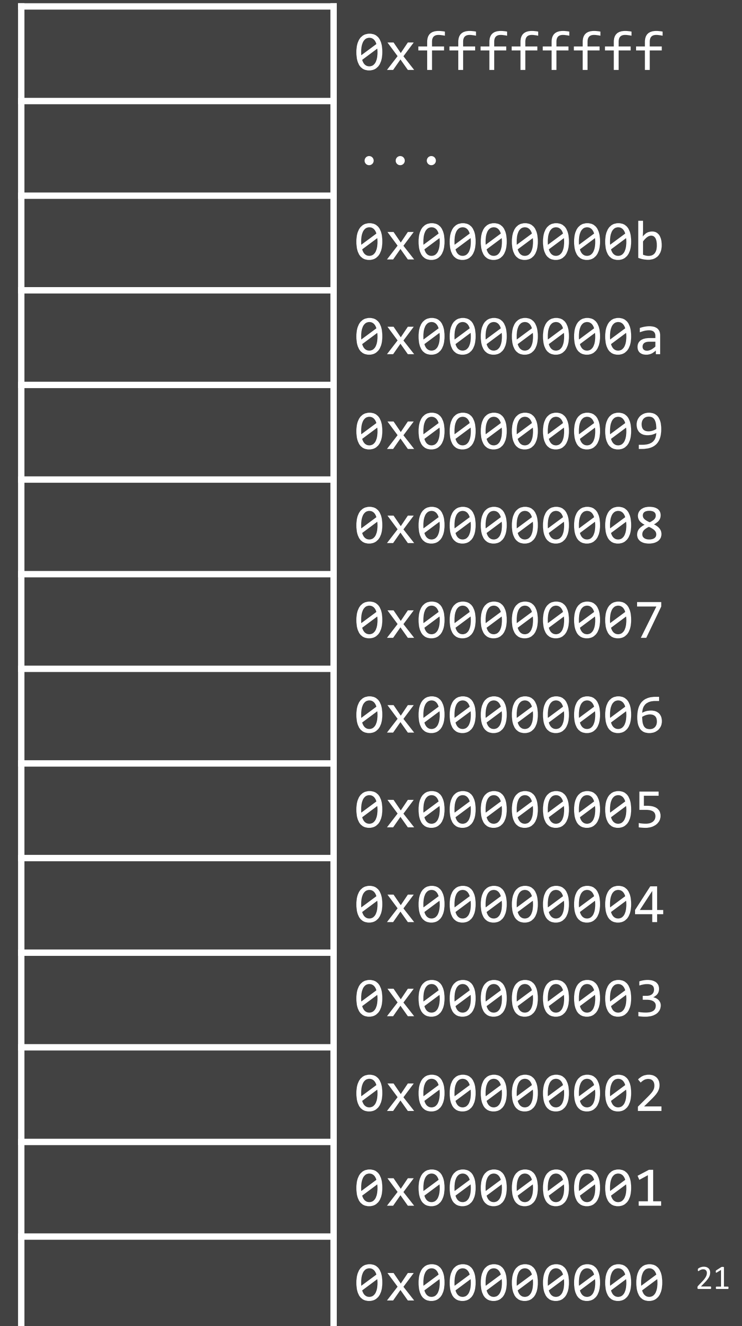
Example:

r5 contains 5 (0x00000005)

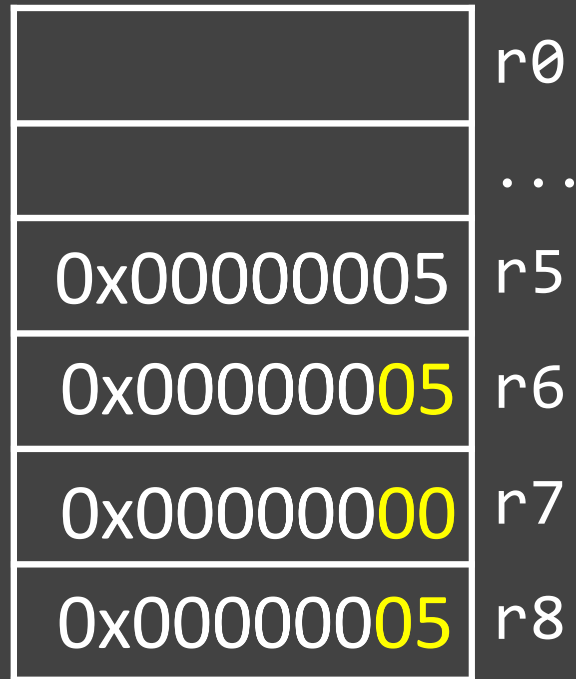
```
SW r5, 8(r0)
```

Clicker Question: After executing the store, which byte address contains the byte 0x05?

a) 0x00000008
b) 0x00000008
c) 0x00000008
d) 0x00000008
e) I don't know



Big Endian Memory Example



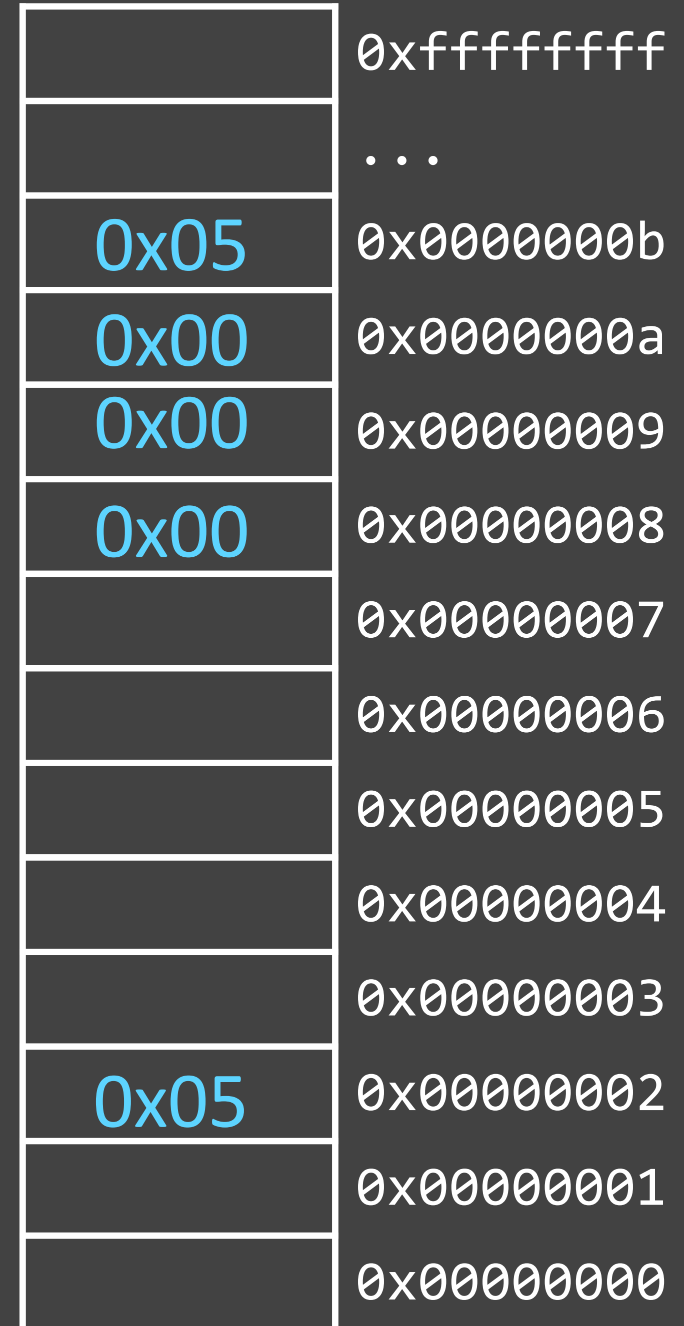
SB r5, 2(r0)

LB r6, 2(r0)

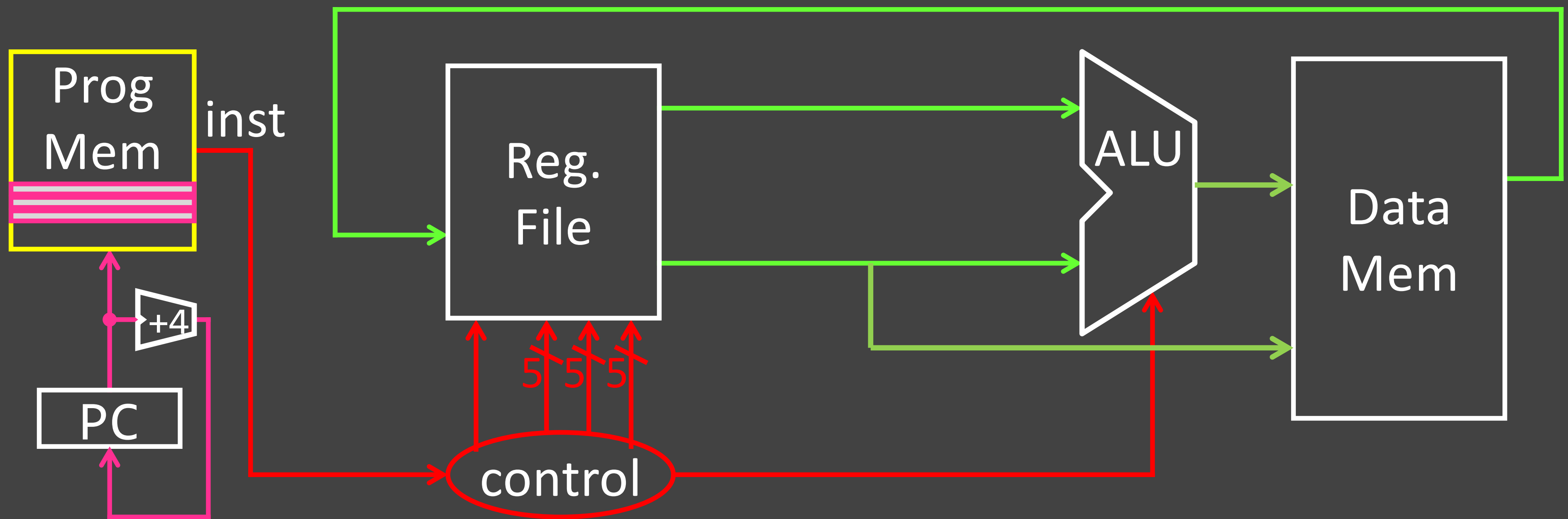
SW r5, 8(r0)

LB r7, 8(r0)

LB r8, 11(r0)



Control Flow



What if the program is more than just a straight line of instructions?

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

000010 01000000000000000000000001

op | immediate

6 | 26 bits

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

"•" = concatenate

$(PC+4)_{31..28}$ | target | 00

4 bits | 26 bits | 2 bits

$(PC+4)_{31..28}$ 01000000000000000000000001 00

MIPS Quirk:

jump targets computed using *already incremented* PC

R-Type (3): Jump Register

00000000011000000000000000001000

op	rs	-	-	-	func
6	5	5	5	5	6 bits

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

Example: JR r3

iClicker Question

What is a good trait about the Jump Register instruction?

(A) Since registers are 32 bits, you can specify any address.
(B) The address you're jumping to is programmable. It doesn't have to be hard-coded in the instruction because it lives in a register.
(C) It allows you to jump to an instruction with an address ending in something other than 00, which is very useful.
(D) Both A and B.
(E) A, B, and C.

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

000100 00101 00001 0000000000000011

op	rs	rd	offset
6	5	5	16 bits

offset ← signed

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)

Example: BEQ r5, r1, 3

```
if(R[r5]==R[r1])
     PC = PC+4 + 12   (i.e. 12 == 3<<2)
```

A word about all these +'s…

I-Type (5): Conditional Jumps

```
000001 00101 00001 0000000000000010
op     rs    subop offset
```

op	rs	subop	offset
6 bits	5 bits	5 bits	16 bits

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)

Example: BGEZ r5, 2

```
if(R[r5] ≥ 0)
     PC = PC+4 + 8  (i.e. 8 == 2<<2)
```

J-Type (2): Jump and Link

00001101000000000000000000000001

op — immediate

6 bits — 26 bits

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

Discuss later

Why?

Function/procedure calls

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

We now know the data path for the MIPS ISA:

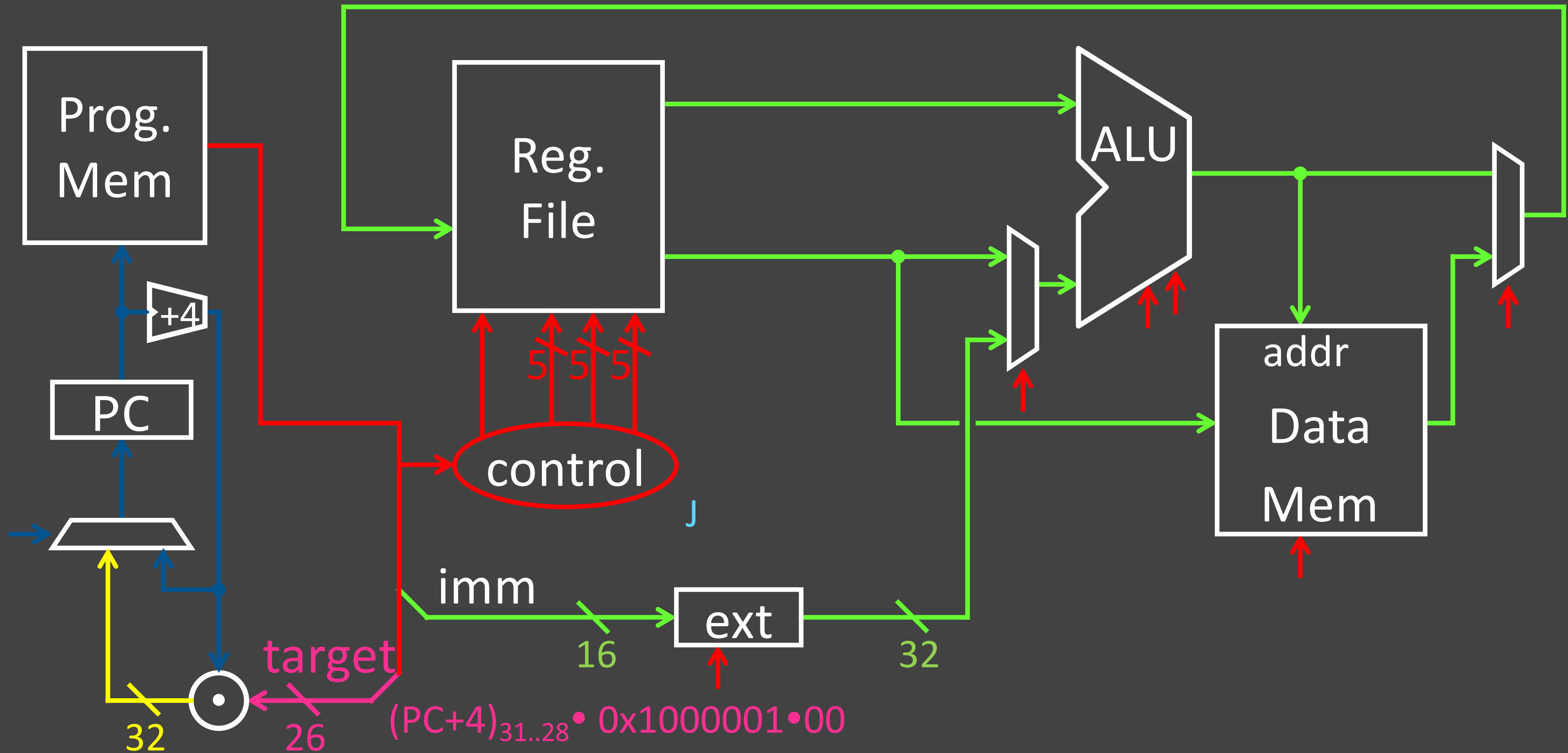
- register, memory and control instructions

Control Instruction Implemenation

You will not need to implement control instructions in Logisim this semester, but if you're curious to know how they are implemented, here are the corresponding slides.

Note: you are still responsible for knowing how to use loads, stores, and control instructions in MIPS assembly.

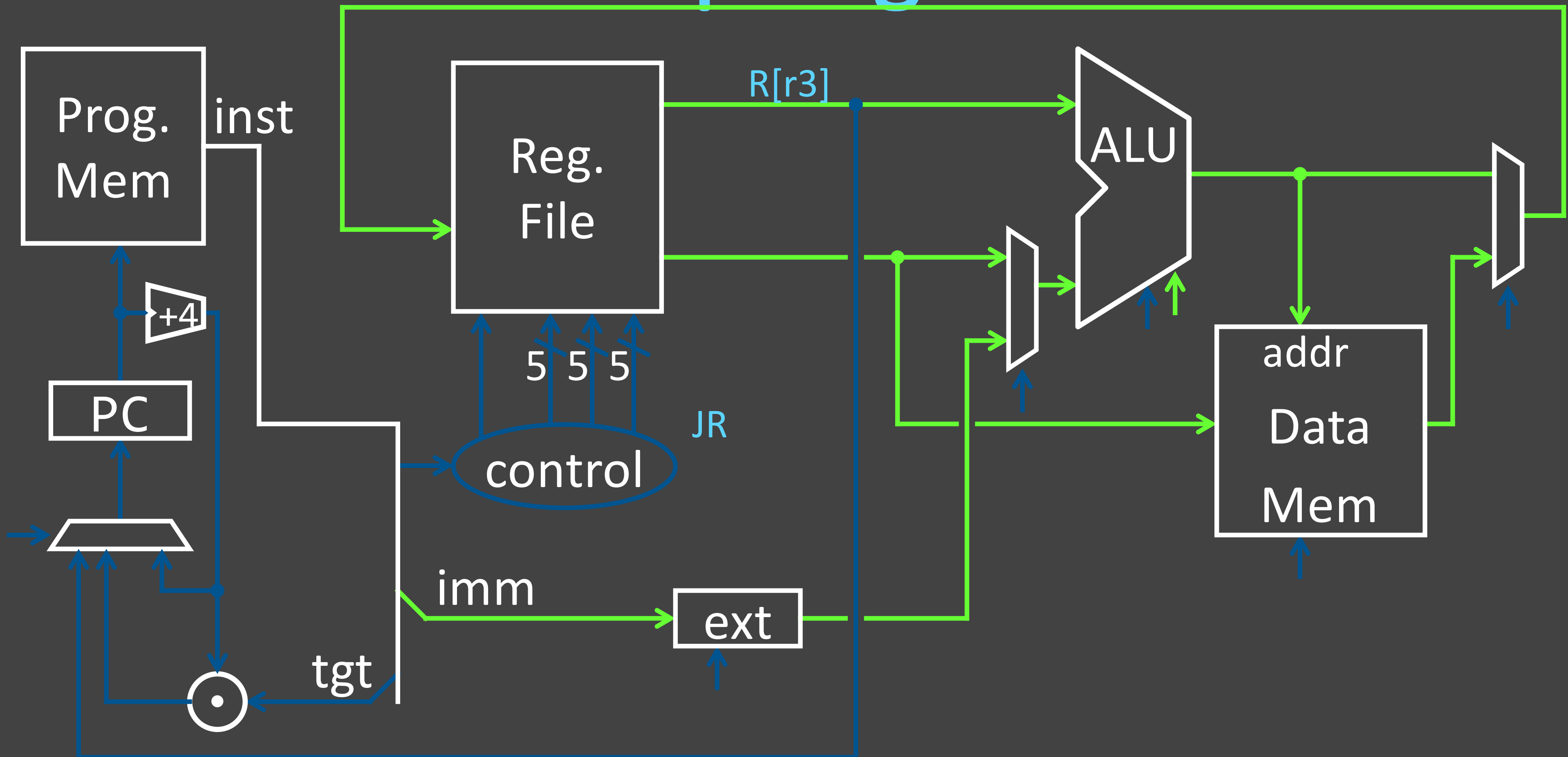
Absolute Jump



(PC+4)$_{31..28}$•0x4000004

Example: PC = (PC+4)$_{31..28}$ • target • 00 # J 0x1000001

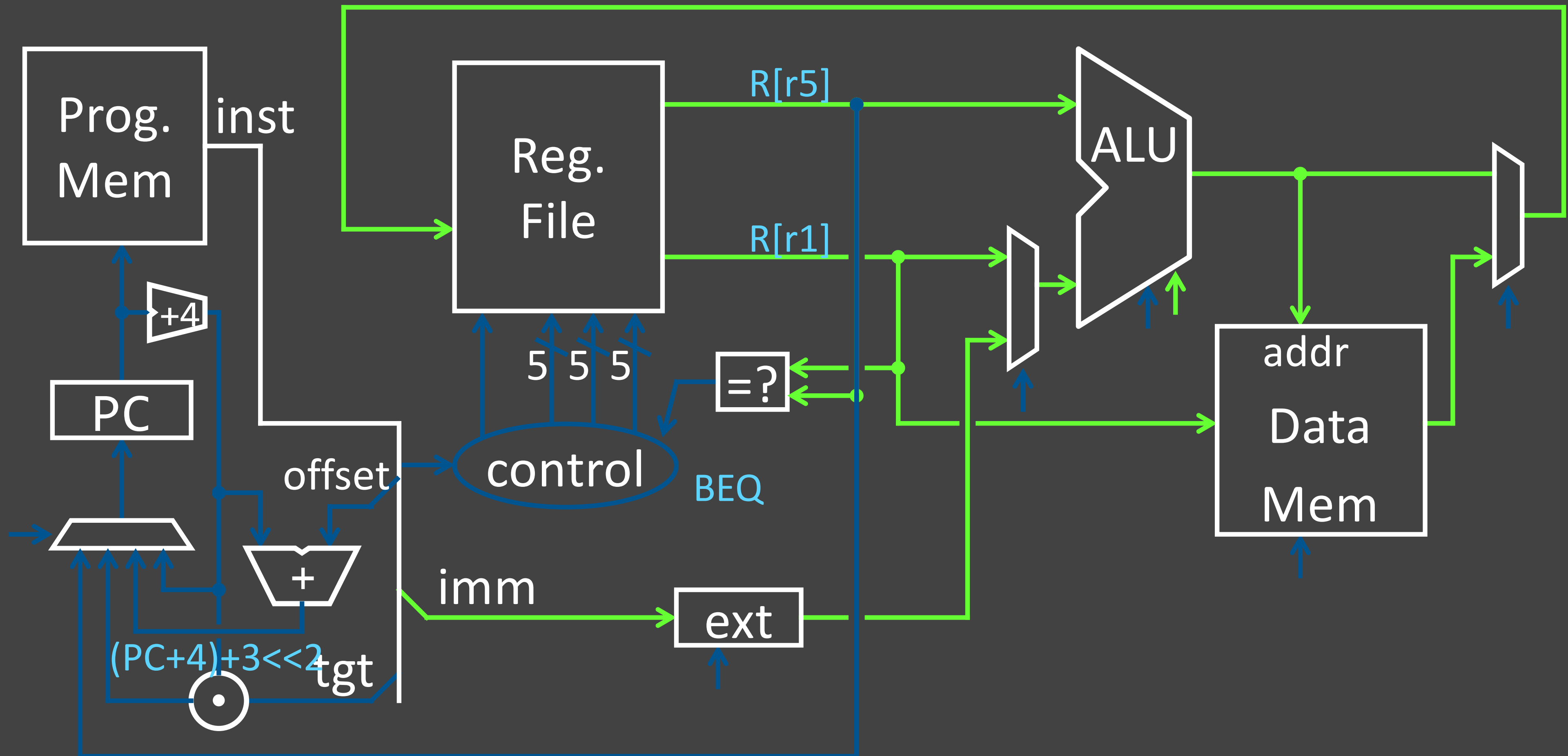
Jump Register



ex: JR r3

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

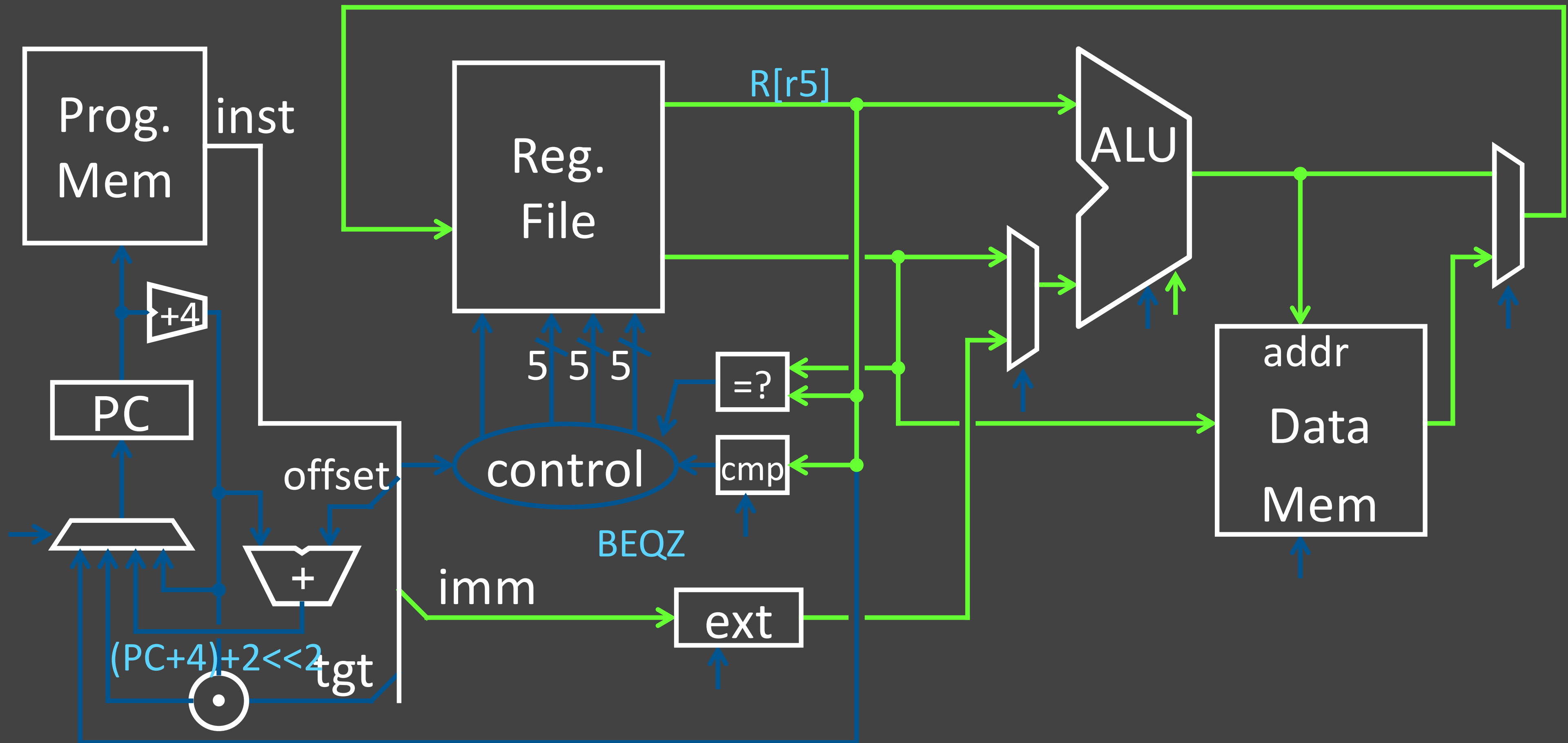
Control Flow: Branches



ex: BEQ r5, r1, 3

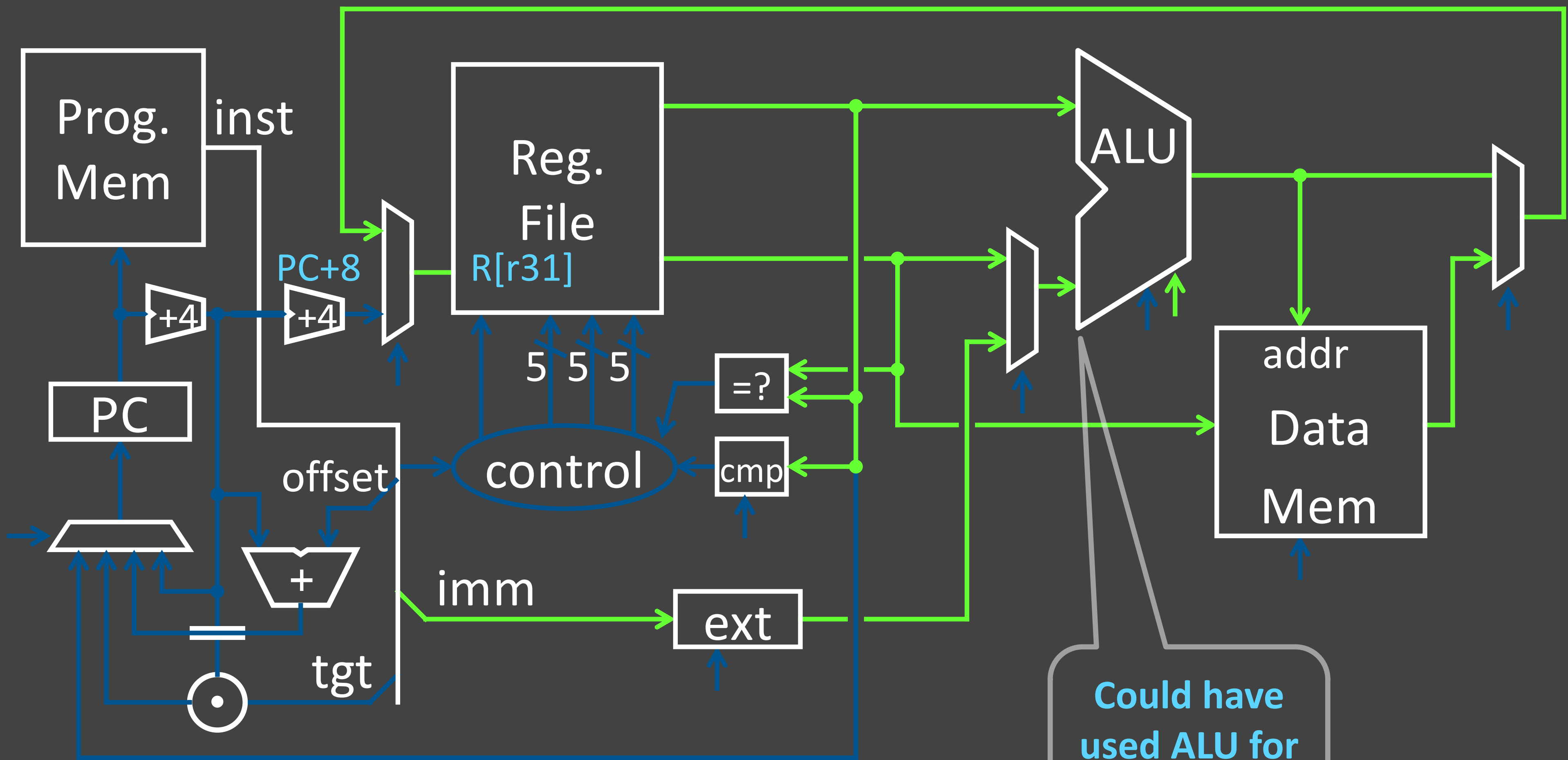
op	mnemonic	description
0x4	BEQ rs, rd, offset	if R[rs] == R[rd] then PC = PC+4 + (offset<<2)

Control Flow: More Branches



op	subop	mnemonic	description
0x1	0x1	BGEZ rs, offset	if R[rs] ≥ 0 then PC = PC+4+ (offset<<2)

Jump and Link



ex: JAL 0x1000001 r31 = PC+8

PC = $(PC+4)_{31..28}$ • 0x4000004

op	mnemonic	description
0x3	JAL target	r31 = PC+8 (+8 due to branch delay slot) PC = $(PC+4)_{31..28}$ • (target << 2)