A Processor

Hakim Weatherspoon
CS 3410, Spring 2010
Computer Science
Cornell University

See: P&H Chapter 2.16-20, 4.1-3

Announcements

HW2 available later today

HW2 due in one week and a half

Work alone

Use your resources

- FAQ, class notes, book, Sections, office hours, newsgroup, CSUGLab

Make sure you

- Registered for class, can access CMS, have a Section, and have a project partner
- Check online syllabus/schedule, review slides and lecture notes, Office Hours, early homework and programming assignments

Announcements

Prelims: ~~Evening~~ of Thursday, March 10 and April 28th

Late Policy

1) Each person has a total of four “slip days”

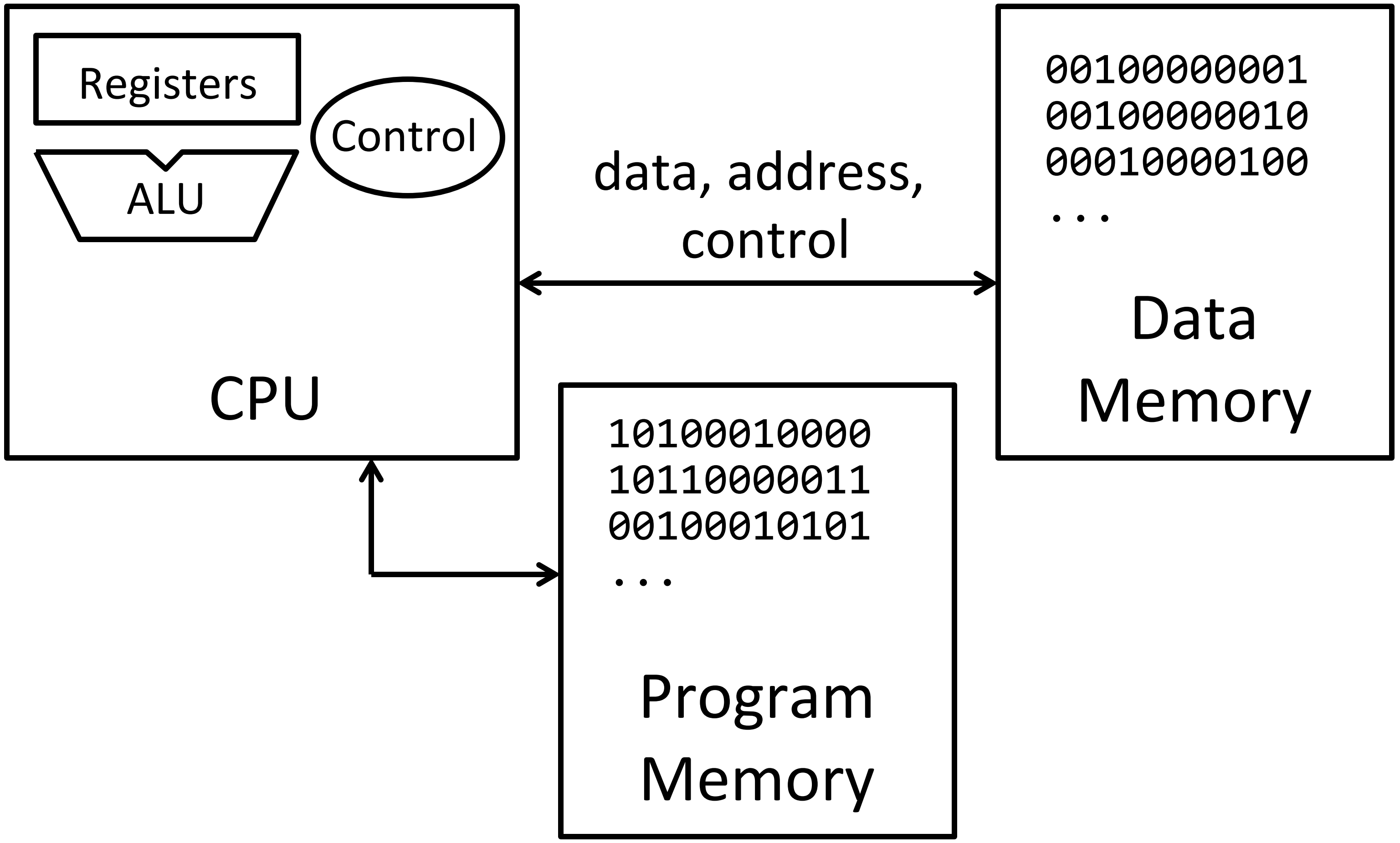
2) For projects, slip days are deducted from all partners

3) 10% deducted per day late after slip days are exhausted

Basic Computer System

Let's build a MIPS CPU

- …but using (modified) Harvard architecture



Instructions

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

High Level Language

- C, Java, Python, Ruby, …
- 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

```
00100000000000100000000000001010
00100000000000010000000000000000
00000000001000100001100000101010
```

Machine Langauge

- Binary-encoded assembly
- Labels become addresses

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

Complexity

MIPS = Reduced Instruction Set Computer (RlSC)

- ≈ 200 instructions, 32 bits each, 3 formats
  - mostly orthogonal
- all operands in registers
  - almost all are 32 bits each, can be used interchangeably
- ≈ 1 addressing mode: Mem[reg + imm]

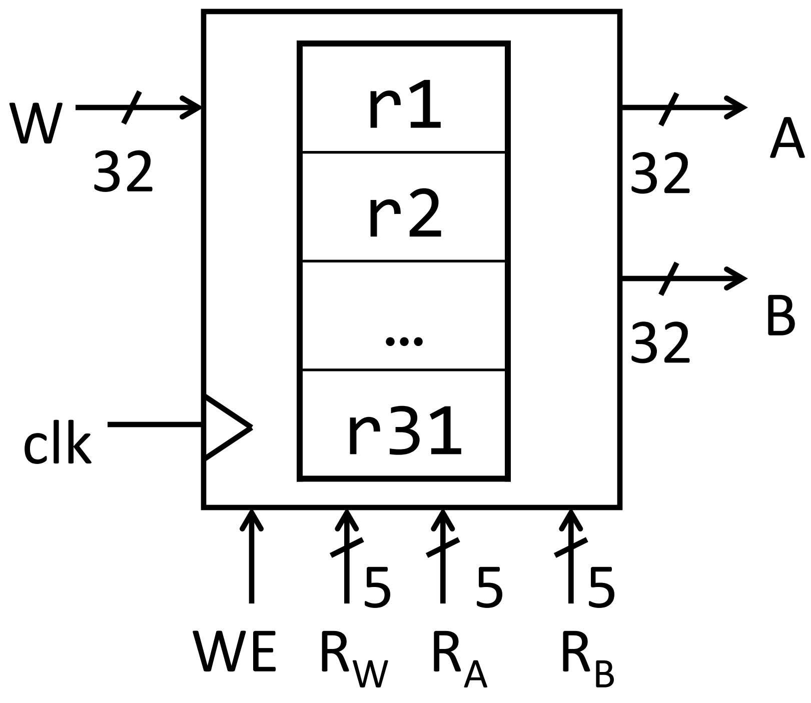
x86 = Complex Instruction Set Computer (ClSC)

- > 1000 instructions, 1 to 15 bytes each
- operands in special 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]

MIPS Register file

MIPS register file

- 32 registers, 32-bits each (with r0 wired to zero)
- Write port indexed via R_W
  - Writes occur on falling edge but only if WE is high
- Read ports indexed via R_A, R_B



MIPS Memory

MIPS Memory

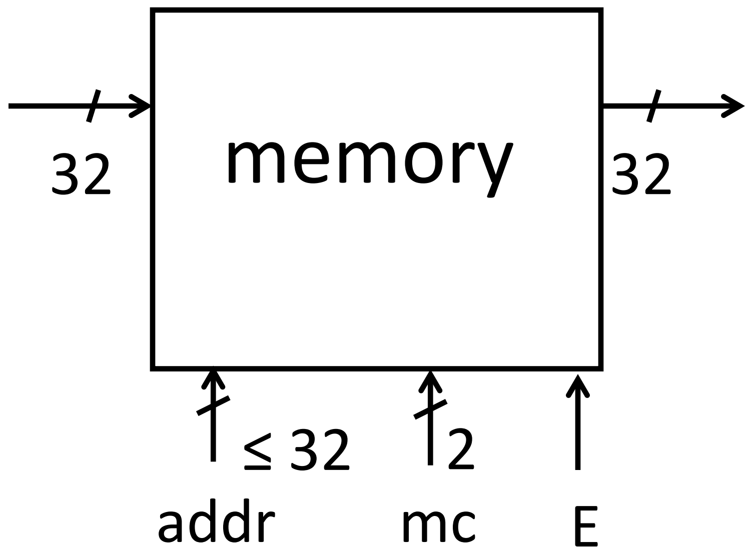
- Up to 32-bit address
- 32-bit data
  (but byte addressed)
- Enable + 2 bit memory control

  00: read word (4 byte aligned)

  01: write byte

  10: write halfword (2 byte aligned)

  11: write word (4 byte aligned)



Instruction Usage

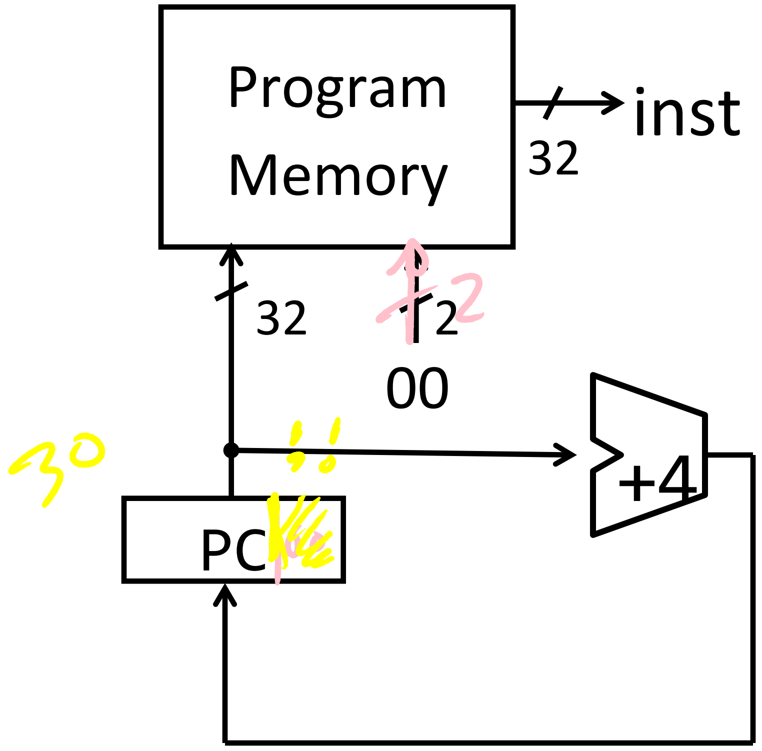
Basic CPU execution loop

1. fetch one instruction
2. increment PC
3. decode
4. execute

Instruction Fetch

Instruction Fetch Circuit

- Fetch instruction from memory
- Calculate address of next instruction
- Repeat



Arithmetic Instructions

```
00000001000000110001000000100110
```

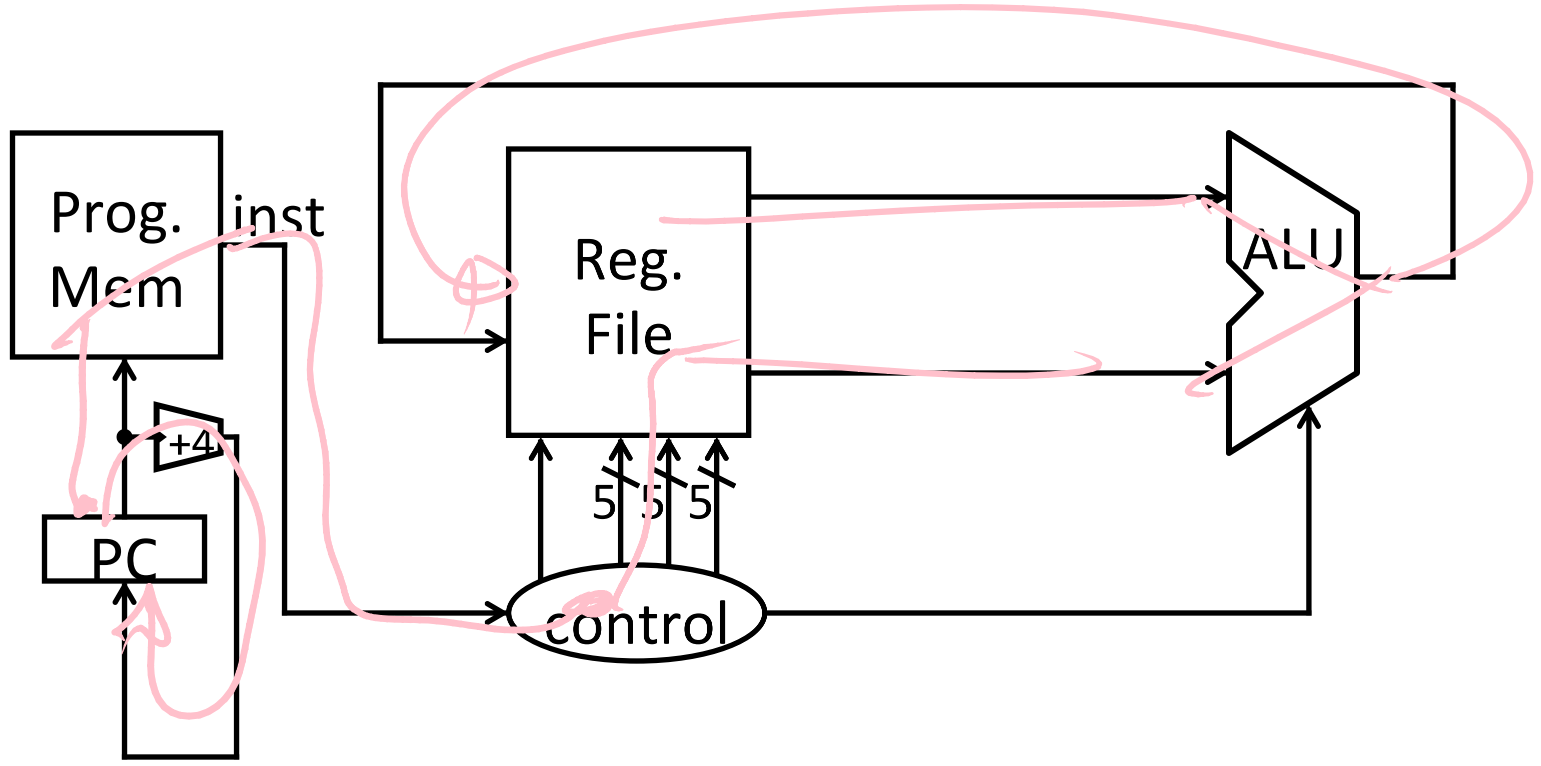
op	rs	rt	rd	-	func
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

0x2 6

R-Type

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] $\oplus$ R[rt]
0x0	0x27	NOR rd, rs rt	R[rd] = ~ (R[rs] | R[rt])

Arithmetic and Logic



Example Programs

r4 = (r1 + r2) | r3

r8 = 4*r3 + r4 – 1

r9 = 9

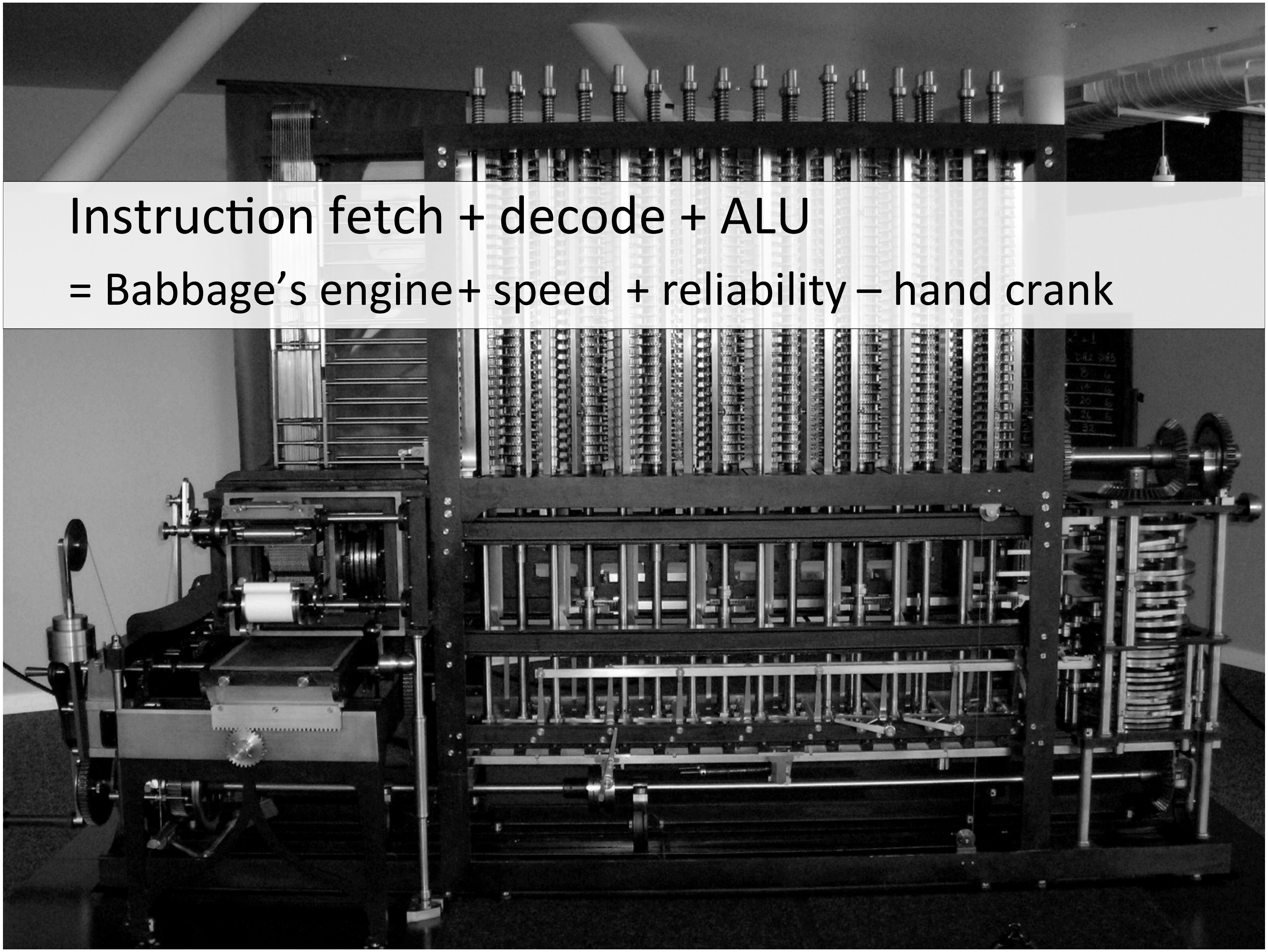
ADDU rd, rs, rt

SUBU rd, rs, rt

OR rd, rs, rt

XOR rd, rs, rt

NOR rd, rs rt



Instruction fetch + decode + ALU

= Babbage's engine+ speed + reliability – hand crank

Arithmetic Instructions: Shift

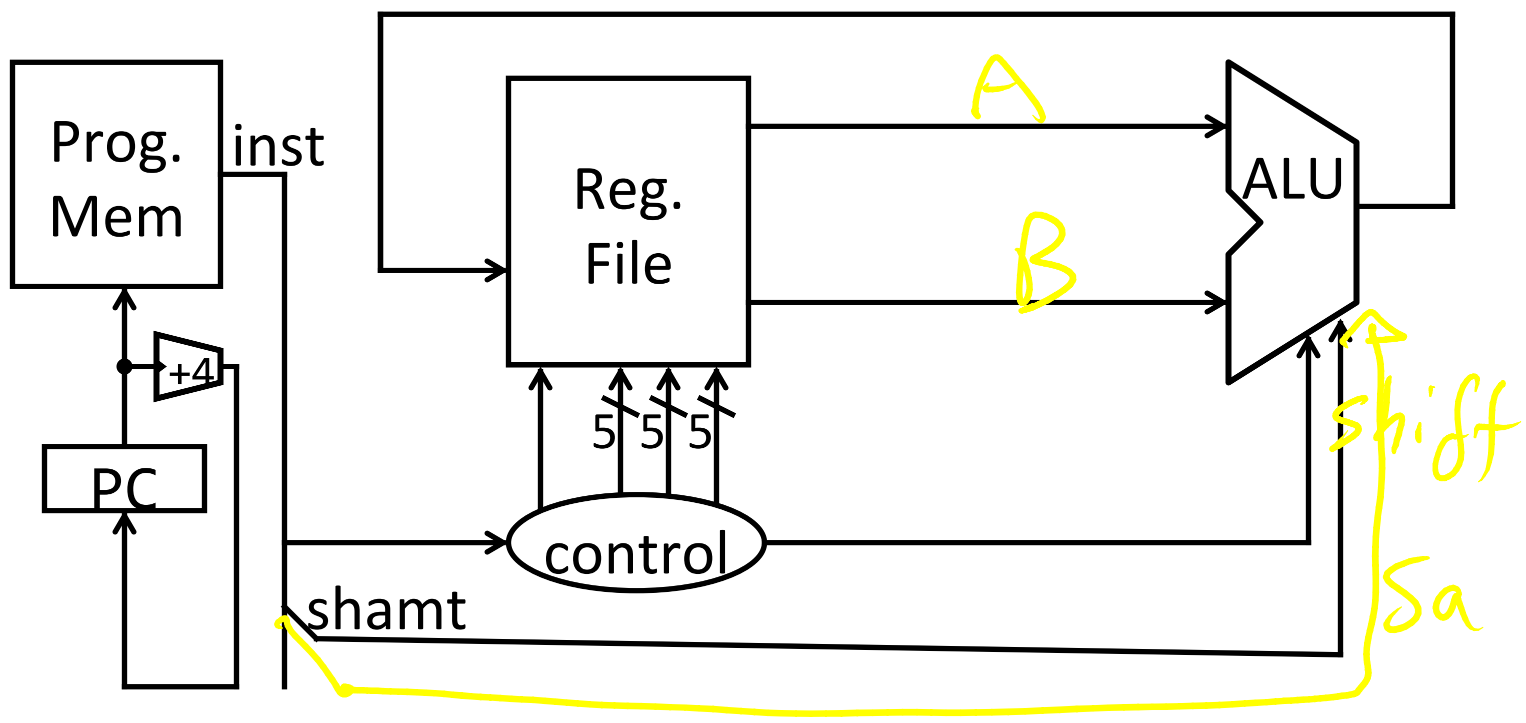
```
00000000000001000100000110000011
|    op    |  -  |  rt  |  rd  | shamt | func |
   6 bits   5 bits 5 bits 5 bits 5 bits  6 bits
```

R-Type

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

ex: r5 = r3 * 8

Shift



r3 = 4 * r5

Arithmetic Instructions: Immediates

```
00100100101001010000000000000101
```

op	rs	rd	immediate
6 bits	5 bits	5 bits	16 bits

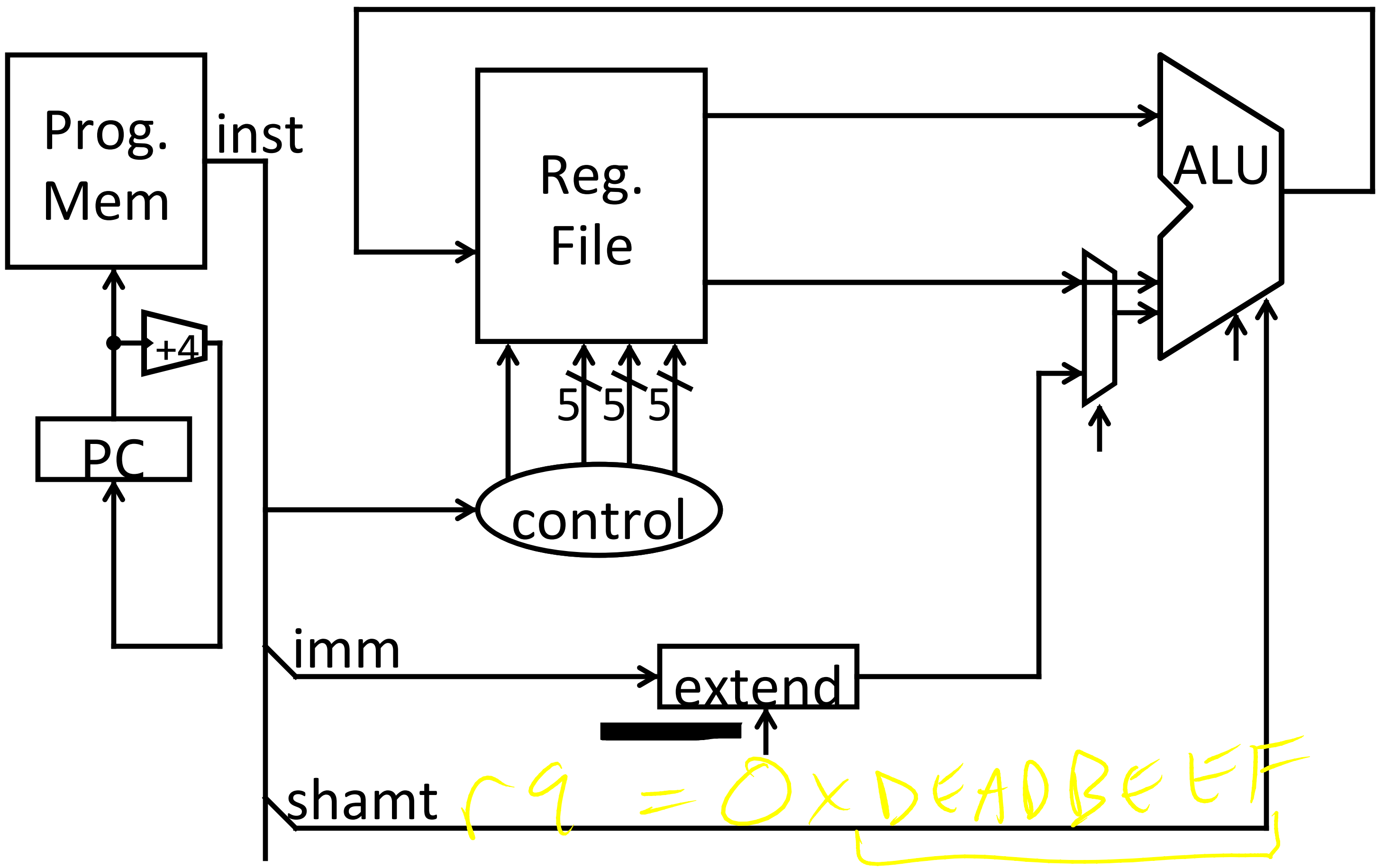
A

I-Type

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)

ex: r5 += 5 ex: r9 = -1 ex: r9 = 65535

Immediates



$r9 = 0x\underbrace{DEADBEEF}$

ORI r9, r0, 0xDEAD
SLL r9, r9, 16
ORI r9, r0, 0xBEEF

Arithmetic Instructions: Immediates

```
00111100000001010000000000000101
```

op	-	rd	immediate
6 bits	5 bits	5 bits	16 bits

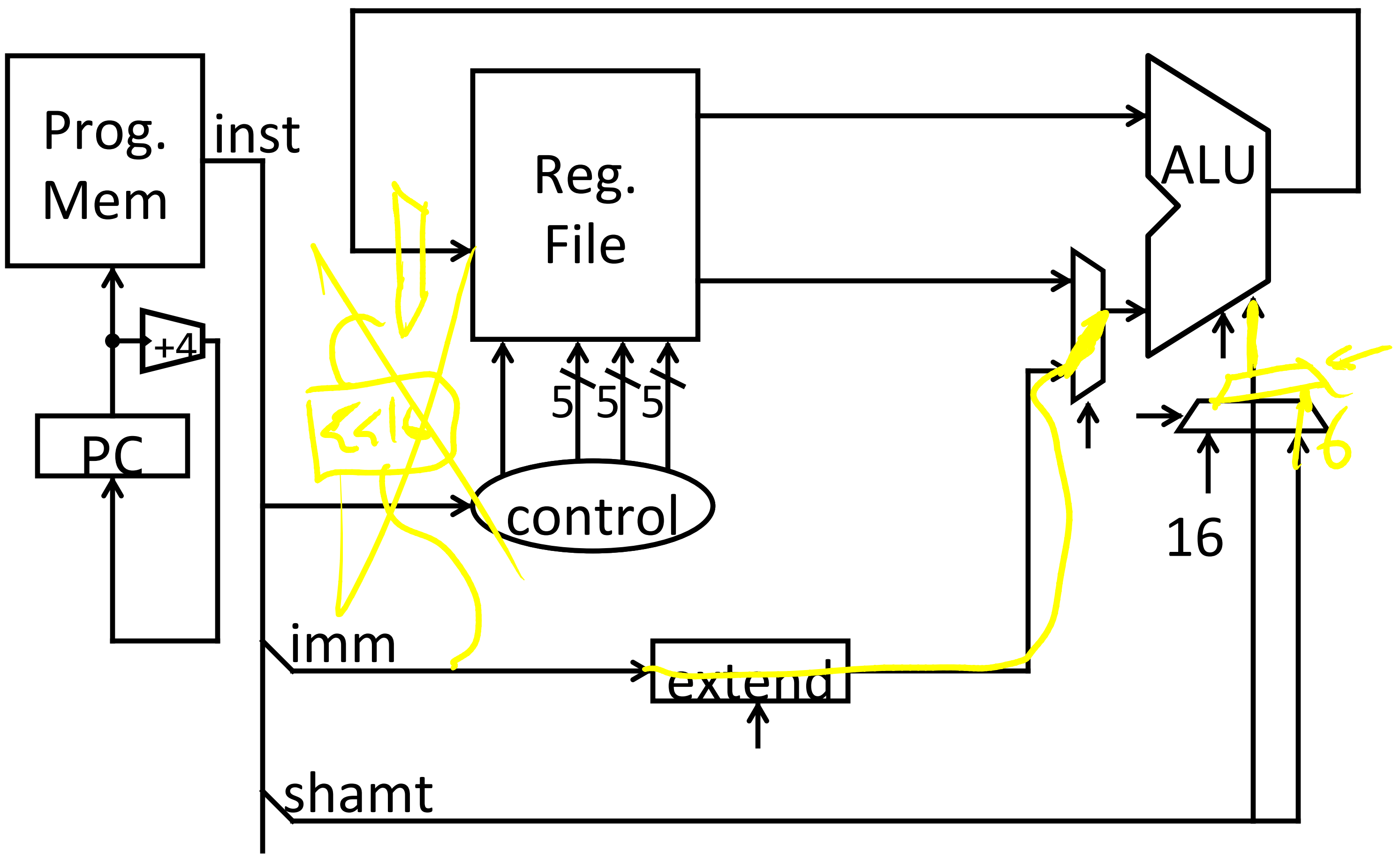
I-Type

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

ex: r5 = 0xdeadbeef

load upper imm.

Immediates



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

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

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

Control flow

- conditional branches: pc-relative addresses
- jumps: fixed offsets, register absolute

Memory Instructions

$$\underbrace{\texttt{101001}}_{\texttt{op}}\underbrace{\texttt{00101}}_{\texttt{rs}}\underbrace{\texttt{00001}}_{\texttt{rd}}\underbrace{\texttt{0000000000000010}}_{\texttt{offset}}$$

op	rs	rd	offset
6 bits	5 bits	5 bits	16 bits

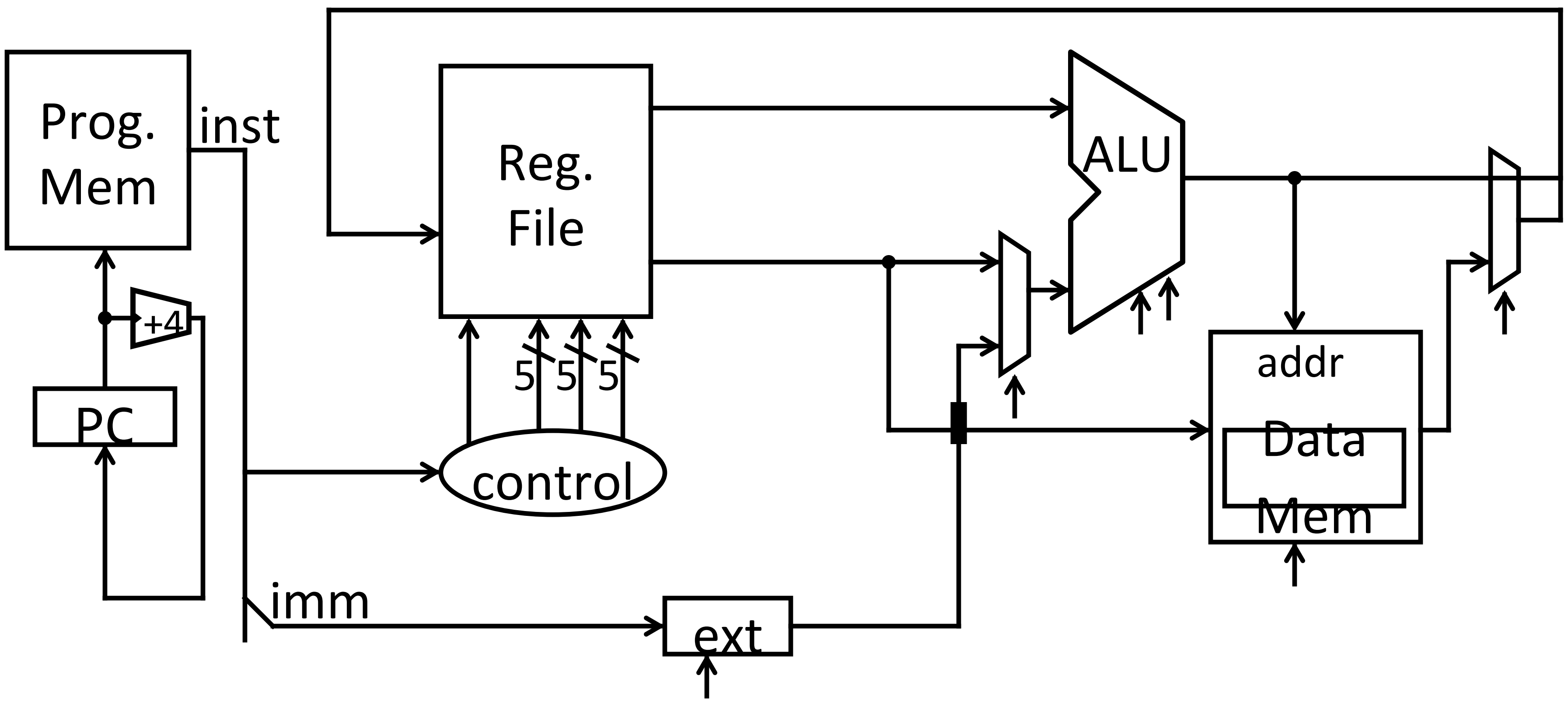
I-Type

base + offset addressing

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]

signed offsets

Memory Operations

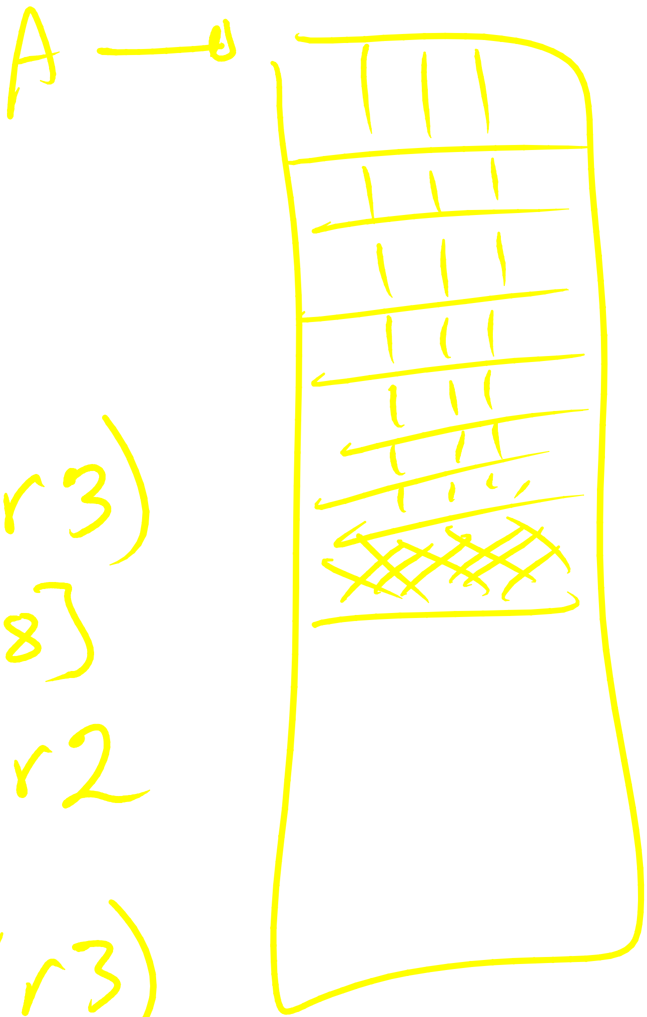


Example

r2 r3

```
int h, A[];
A[12] = h + A[8];
```

LW r4, 32(r3)
# r4 = A[8]
ADD r5, r4, r2
SW r5, 48(r3)

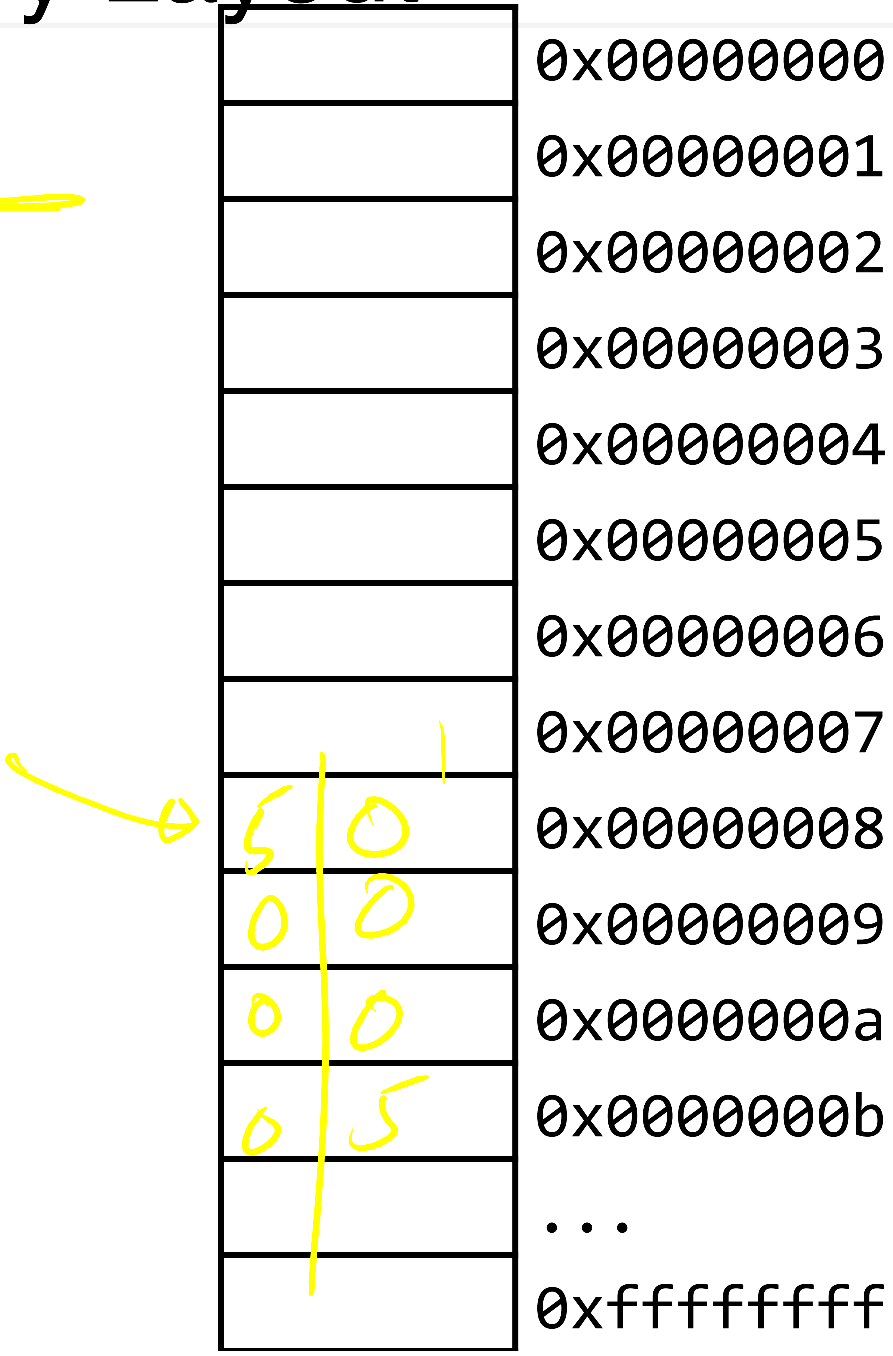


Memory Layout

Examples:

```
# r5 contains 0x5
sb r5, 2(r0)
lb r6, 2(r0)

sw r5, 8(r0)
lb r7, 8(r0)
lb r8, 11(r0)
```



Endianness

Endianness: Ordering of bytes within a memory word

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

	1000	1001	1002	1003
as 4 bytes	78	56	34	12
as 2 halfwords	5678		1234	
as 1 word	0x12345678			

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

	1000	1001	1002	1003
as 4 bytes	12	34	56	78
as 2 halfwords	1234		5678	
as 1 word	0x12345678			

Control Flow: Absolute Jump

$$\underbrace{\texttt{000010}}_{\texttt{op}}\underbrace{\texttt{10100001001000001100000011}}_{\texttt{immediate}}$$

6 bits 26 bits

J-Type

op	**mnemonic**	**description**
0x2	J target	PC = $(PC+4)_{32..29}$ || target || 00

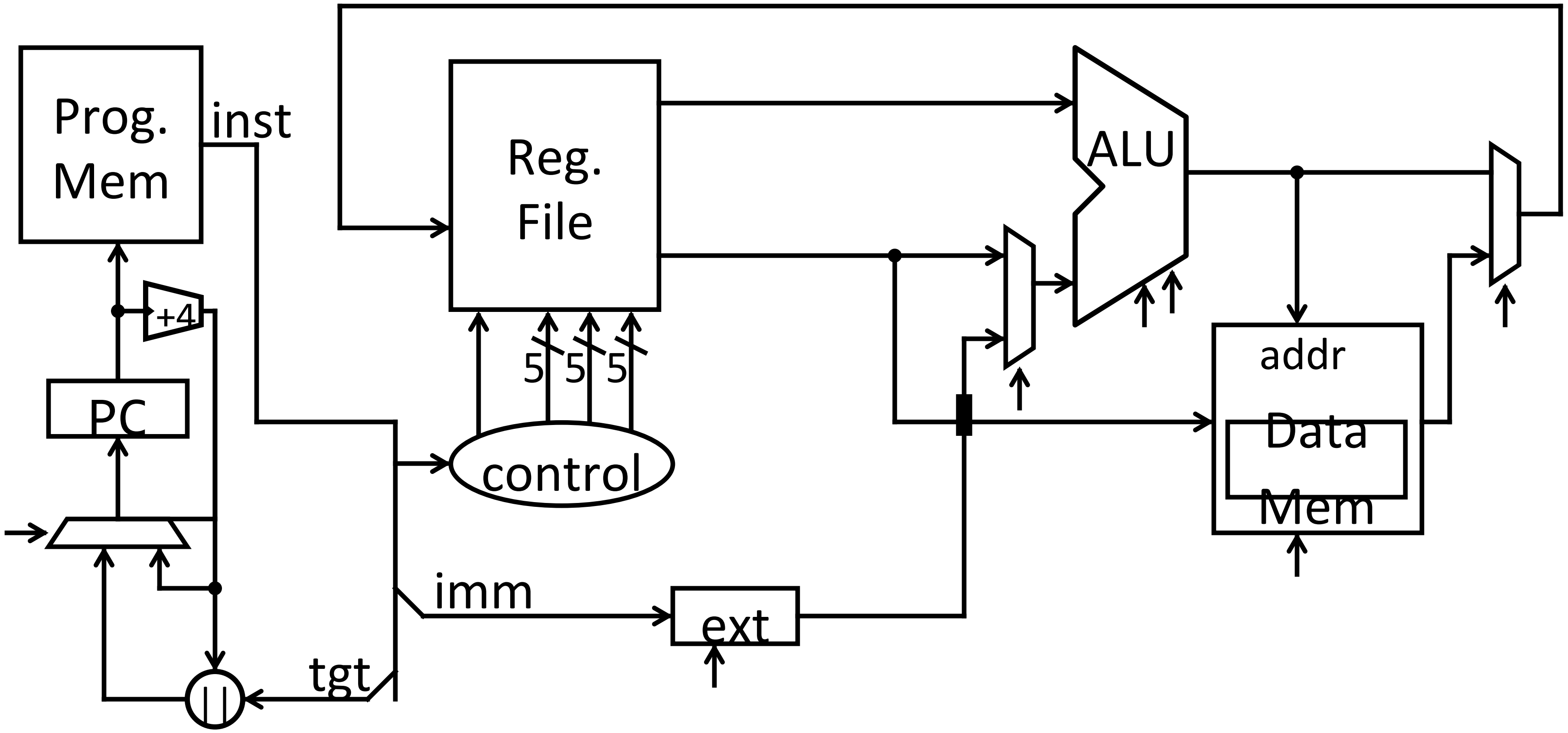
Absolute addressing for jumps

- Jump from 0x30000000 to 0x20000000?
  - But: Jumps from 0x2FFFFFFF to 0x3xxxxxxx are possible, but not reverse
- Trade-off: out-of-region jumps vs. 32-bit instruction encoding

MIPS Quirk:

- jump targets computed using *already incremented* PC

Absolute Jump



Control Flow: Jump Register

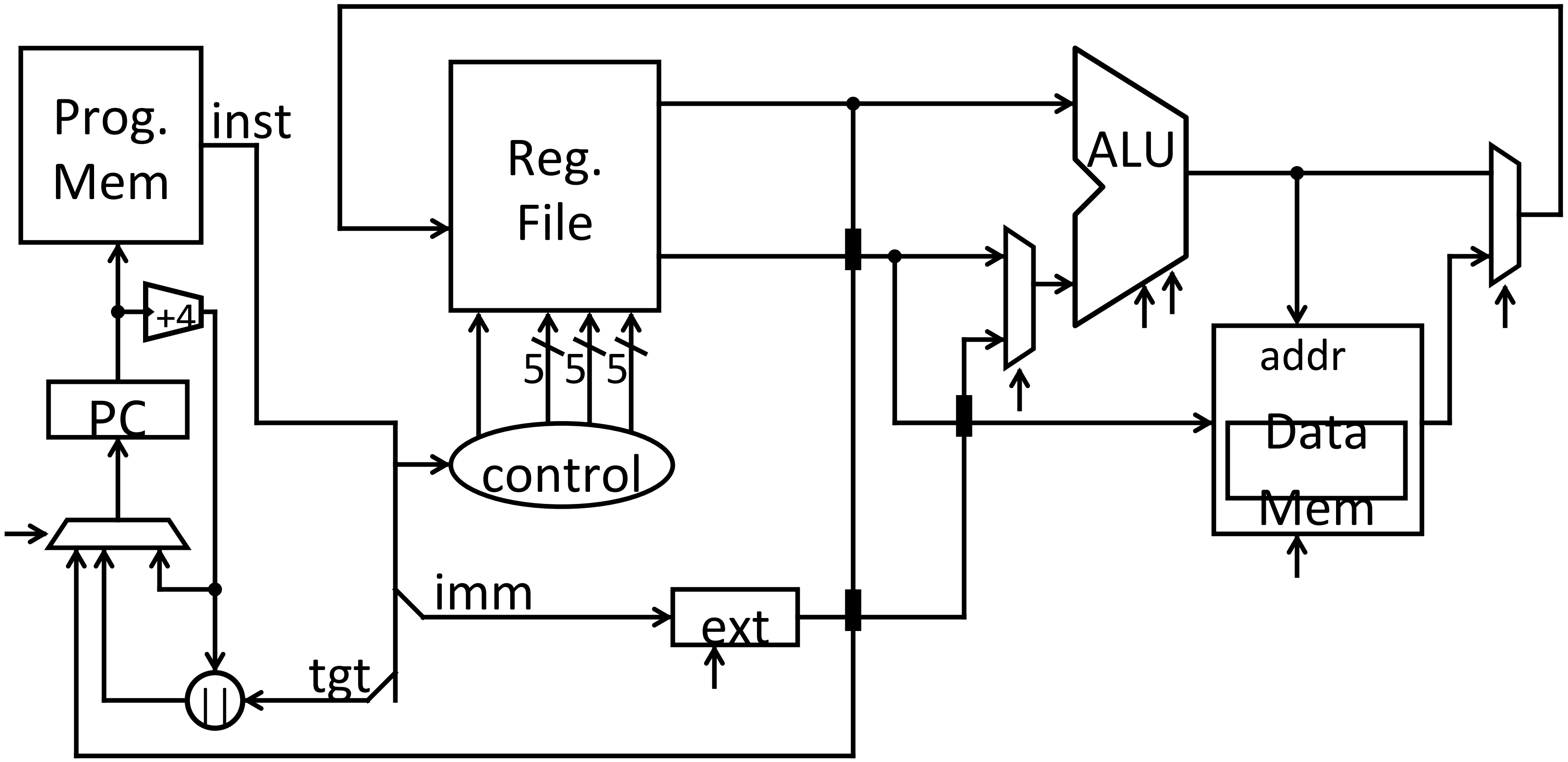
```
00000000011000000000000000001000
```

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

R-Type

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

Jump Register



Examples (2)

jump to 0xabcd1234 45

~~J 0xabcd1234~~

LUI r3, 0xabcd

ORI r3, r3, 0xf2345

JR r3

[redacted]

Examples (2)

```
# assume 0 <= r3 <= 1
if (r3 == 0) jump to 0xdecafe0
else jump to 0xabcd1234
```

= X

= Y

r5 = Y - X

r5 = X + r3 * r5

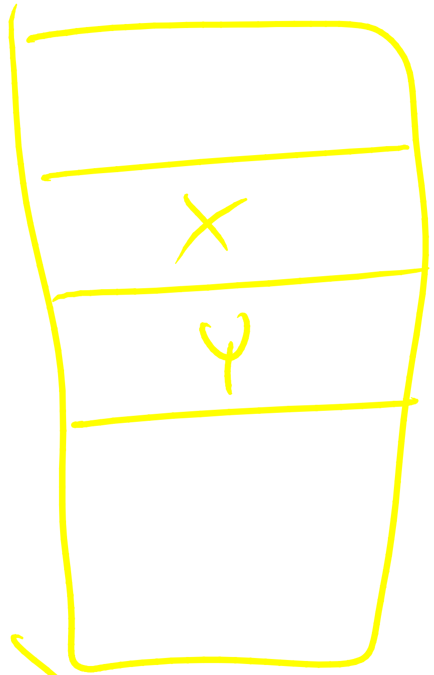
JR r5

[redacted]

Examples (2)

```
# assume 0 <= r3 <= 1
if (r3 == 0) jump to 0xdecafe0   = x
else jump to 0xabcd1234   = y
```

```
r4 = x
sw r4, 0(r0)
r4 = y
sw r4, 4(r0)
```



Control Flow: Branches

$$\underbrace{000100}_{\text{op}}\underbrace{00101}_{\text{rs}}\underbrace{00001}_{\text{rd}}\underbrace{0000000000000011}_{\text{offset}}$$

op	rs	rd	offset
6 bits	5 bits	5 bits	16 bits

I-Type

signed offsets

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)

Examples (3)

r5 r6

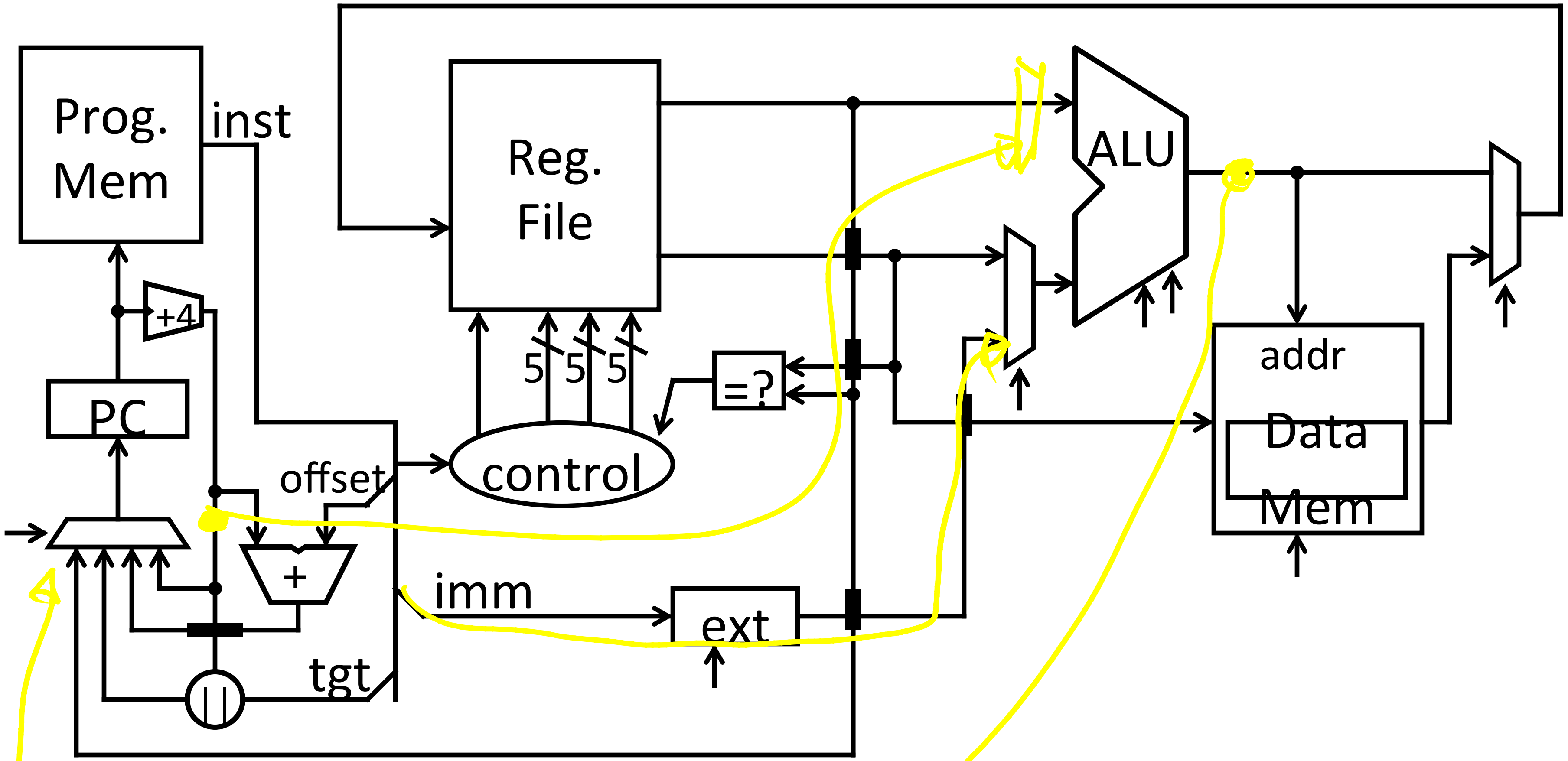
```
if (i == j) { i = i * 4; }
else { j = i - j; }
```

– if

– else

```
BNE r5, r6, ELSE (+8) (2)
    SLL r5, r5, 2
BEQ r0, r0, BOT (+4) (3)
ELSE: SUB r6, r5, r6
BOT:
```

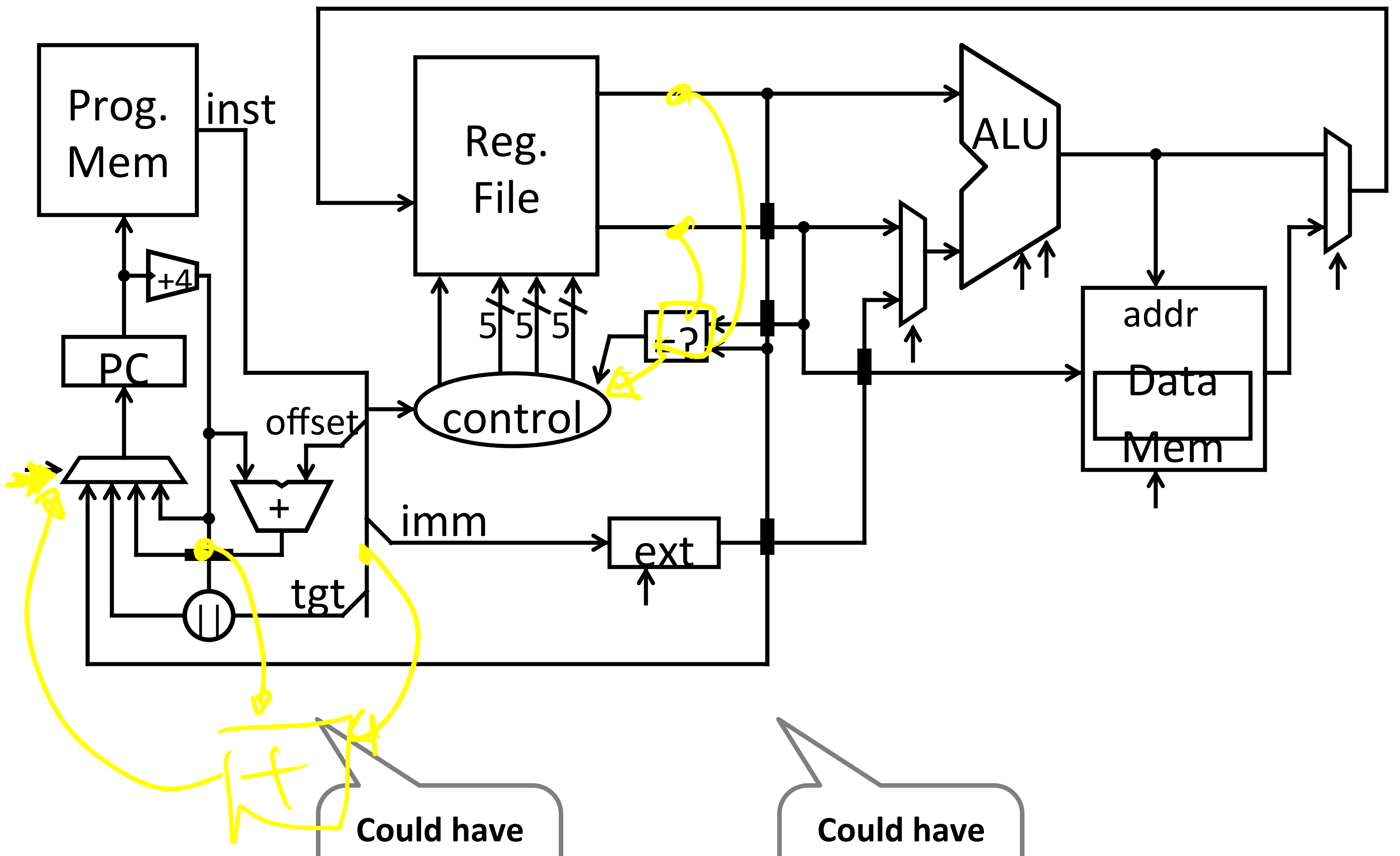
Absolute Jump



Could have used ALU for branch add

Could have used ALU for branch cmp

Absolute Jump



Could have used ALU for branch add

Could have used ALU for branch cmp

Control Flow: More Branches

$\underbrace{000001}_{\text{op}}\underbrace{00101}_{\text{rs}}\underbrace{00001}_{\text{subop}}\underbrace{0000000000000010}_{\text{offset}}$

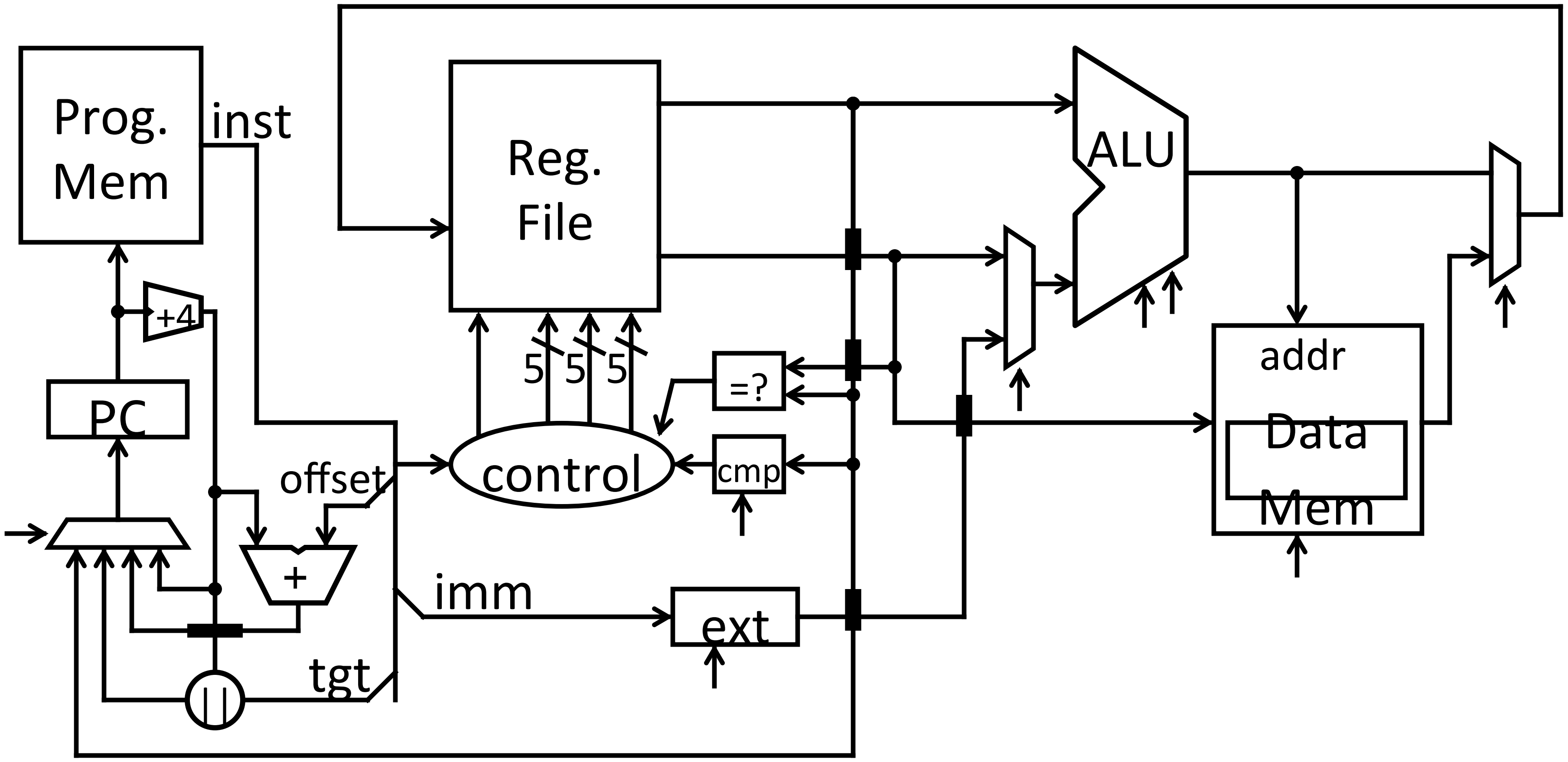
op	rs	subop	offset
6 bits	5 bits	5 bits	16 bits

almost I-Type

signed offsets

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)

Absolute Jump



Could have used ALU for branch cmp

Control Flow: Jump and Link

$$\underbrace{\texttt{000011}}_{\substack{\texttt{op} \\ \texttt{6 bits}}}\underbrace{\texttt{00000001001000011000000010}}_{\substack{\texttt{immediate} \\ \text{26 bits}}}$$

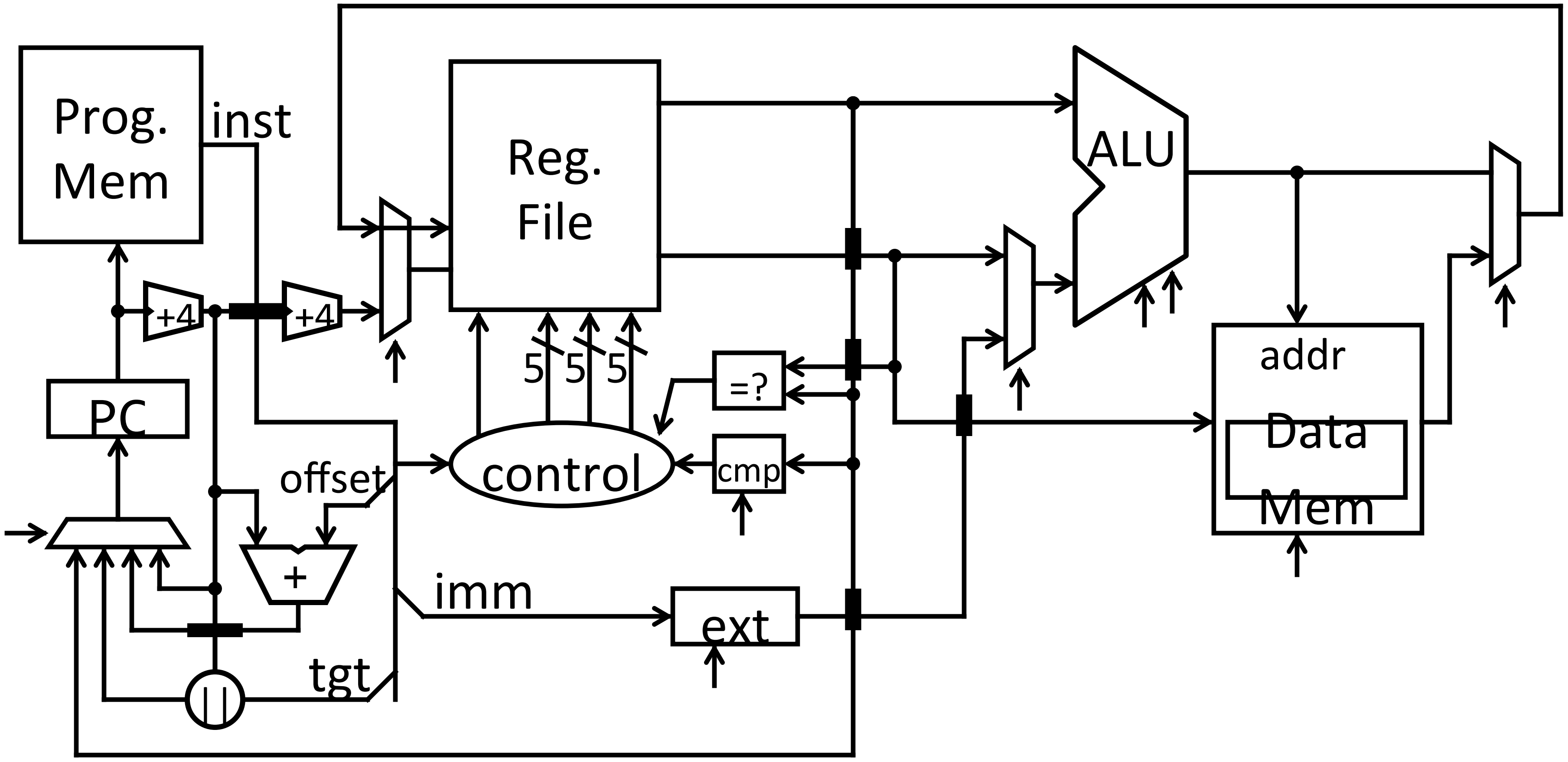
J-Type

op	**mnemonic**	**description**

0x3	JAL target	r31 = PC+8
		PC = $(PC+4)_{32..29}$ || target || 00

J

Absolute Jump



Could have used ALU for link add

Next Time

CPU Performance

Pipelined CPU