

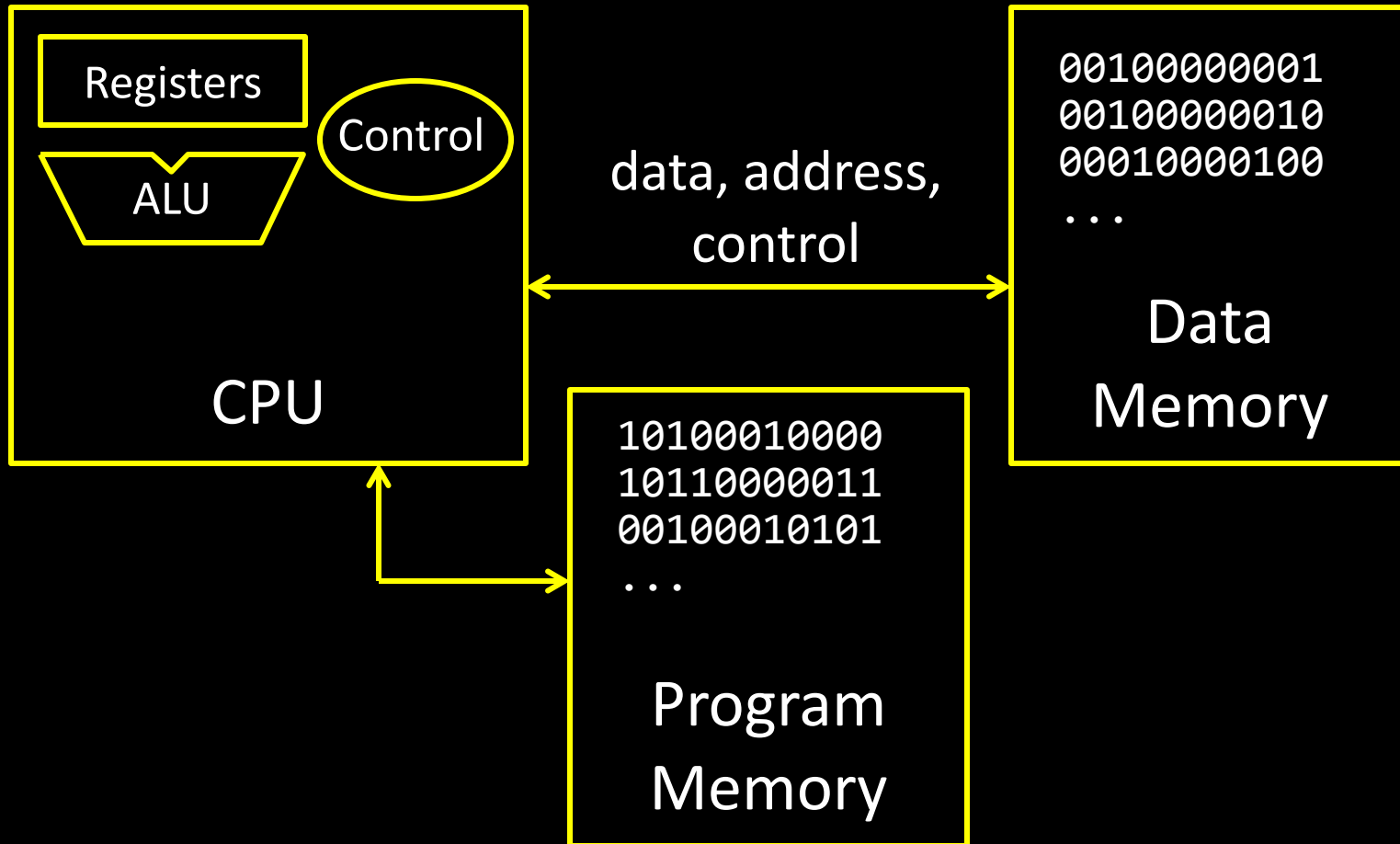
A Processor

Kevin Walsh
CS 3410, Spring 2010
Computer Science
Cornell University

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

Let's build a **MIPS CPU**

- ...but using Harvard architecture



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



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



```
0010000000000110000000000000011
0010000000000101000000000000000
0001000010000101000000000000011
```

High Level Language

- C, Java, Python, Ruby, ...
- Loops, control flow, variables

Assembly Language

- No symbols (except labels)
- One operation per statement

Machine Language

- Binary-encoded assembly
- Labels become addresses

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 = Reduced Instruction Set Computer (RISC)

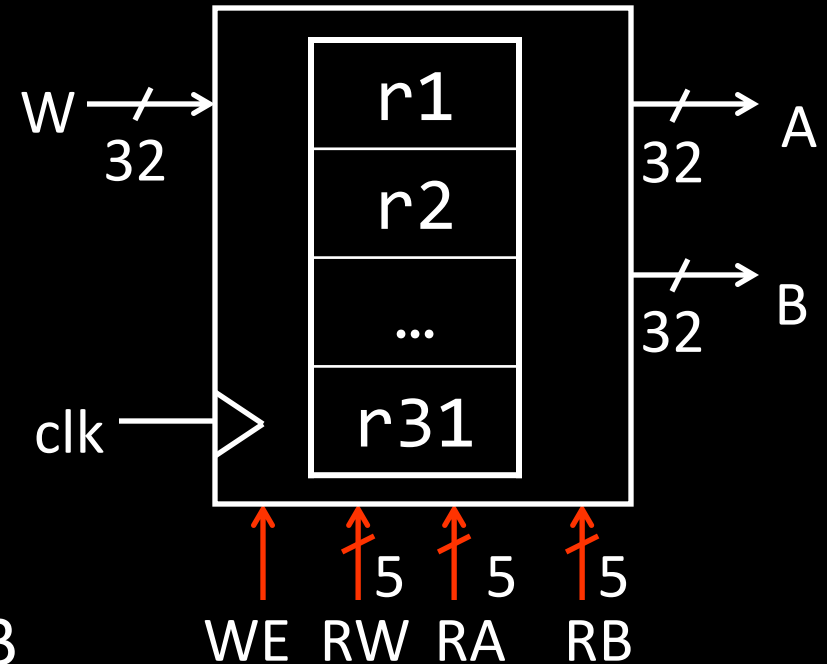
- $\approx$ 200 instructions, 32 bits each, 3 formats
 - mostly orthogonal
- all operands in registers
 - almost all are 32 bits each, can be used interchangeably
- $\approx$ 1 addressing mode: $\text{Mem}[\text{reg} + \text{imm}]$

x86 = Complex Instruction Set Computer (CISC)

- $>$ 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. $\text{Mem}[\text{segment} + \text{reg} + \text{reg} * \text{scale} + \text{offset}]$

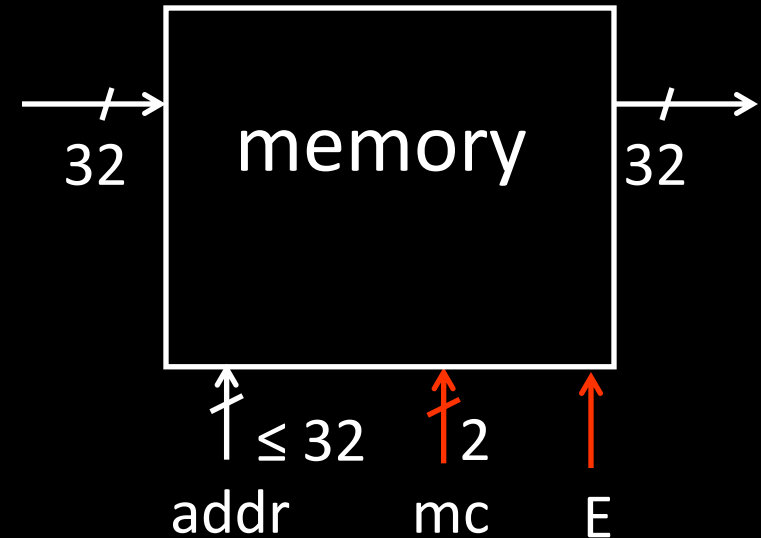
MIPS register file

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



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)

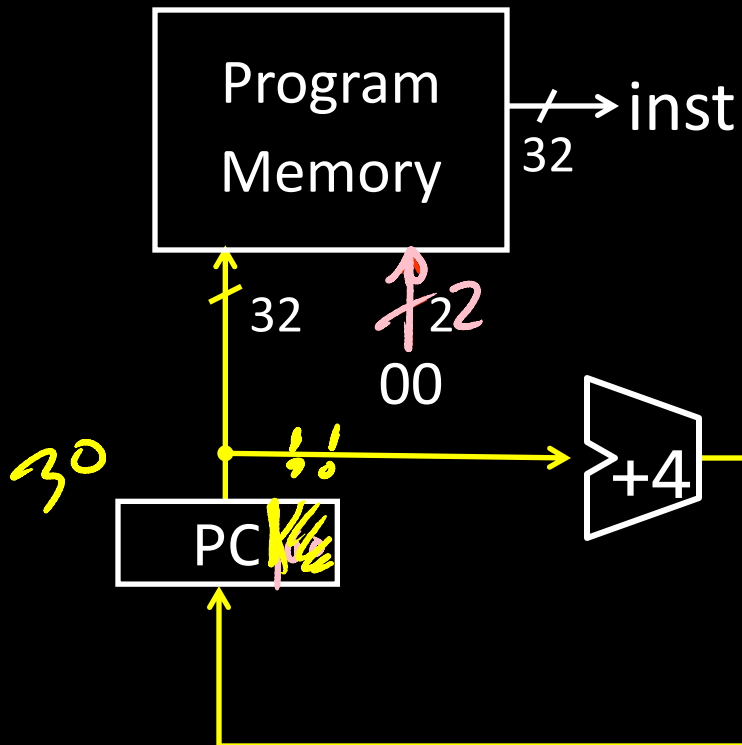


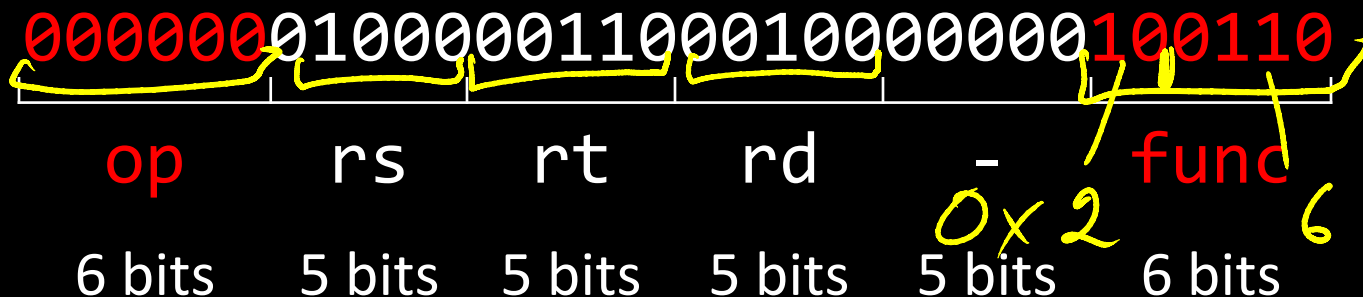
Basic CPU execution loop

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

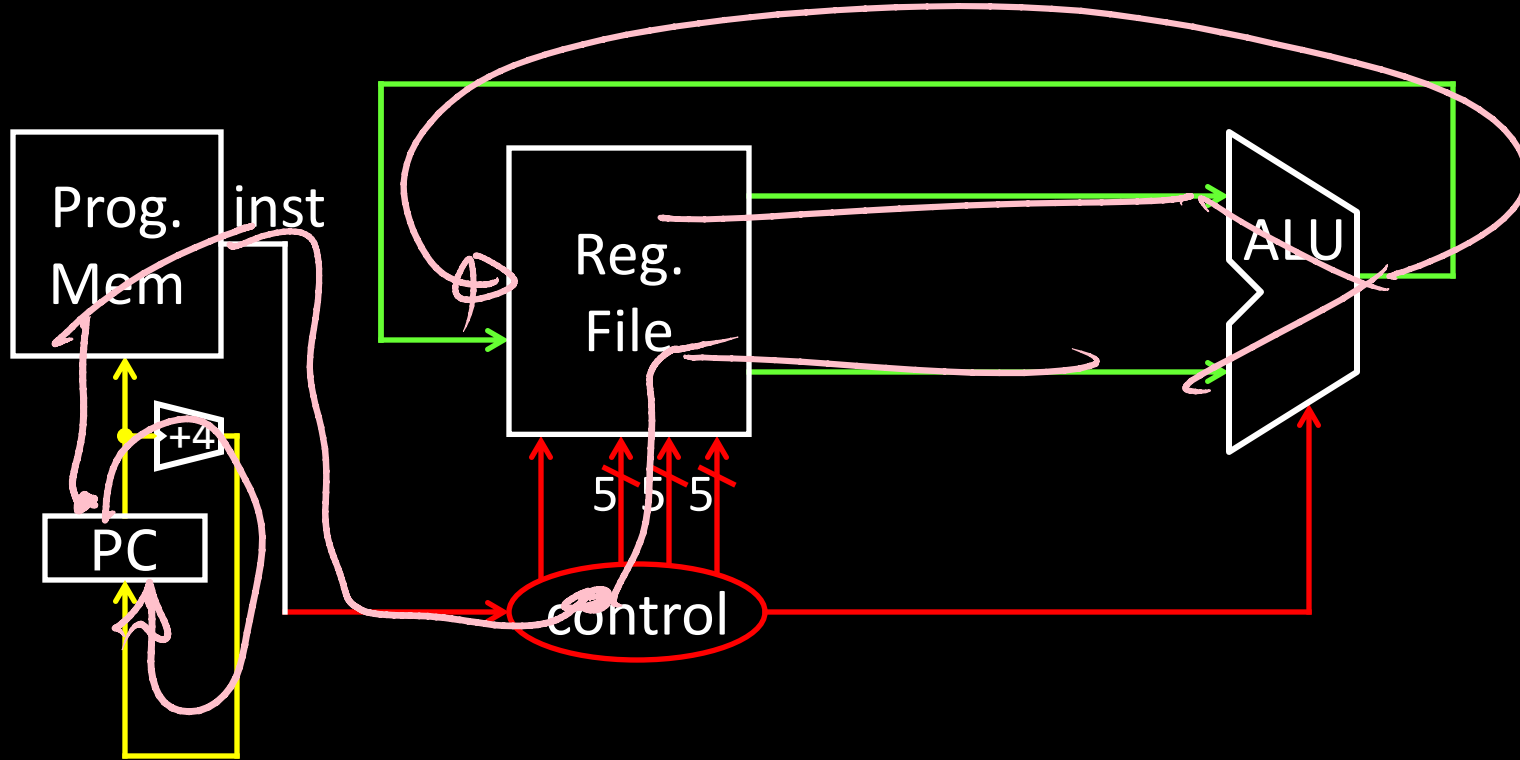
Instruction Fetch Circuit

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





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	$R[rd] = R[rs] \mid R[rt]$
0x0	0x26	XOR rd, rs, rt	$R[rd] = R[rs] \oplus R[rt]$
0x0	0x27	NOR rd, rs rt	$R[rd] = \sim (R[rs] \mid R[rt])$



$$r4 = (r1 + r2) \mid r3$$
$$r8 = 4 * r3 + r4 - 1$$
$$r9 = 9$$

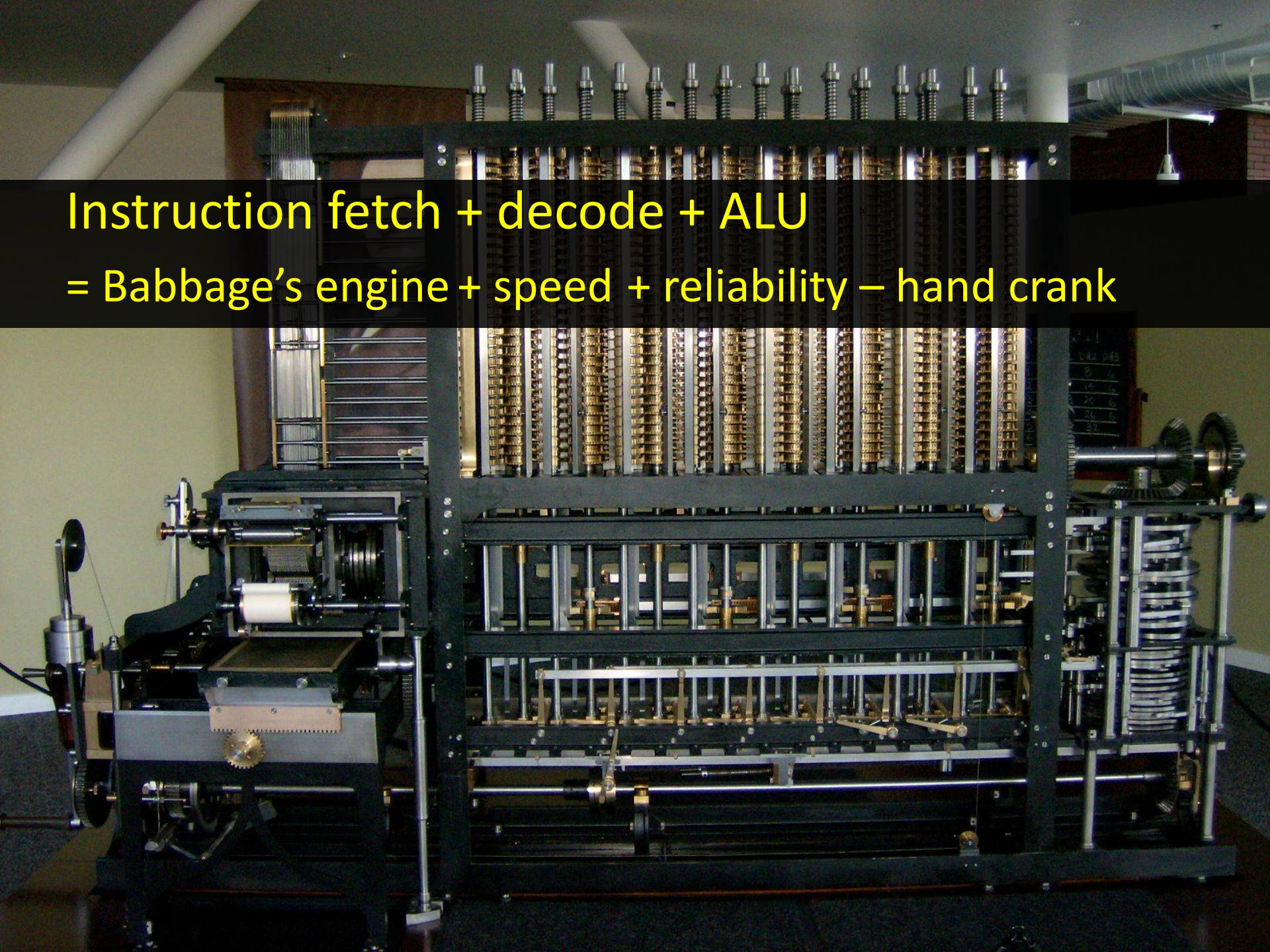
ADDU rd, rs, rt

SUBU rd, rs, rt

OR rd, rs, rt

XOR rd, rs, rt

NOR rd, rs rt



Instruction fetch + decode + ALU

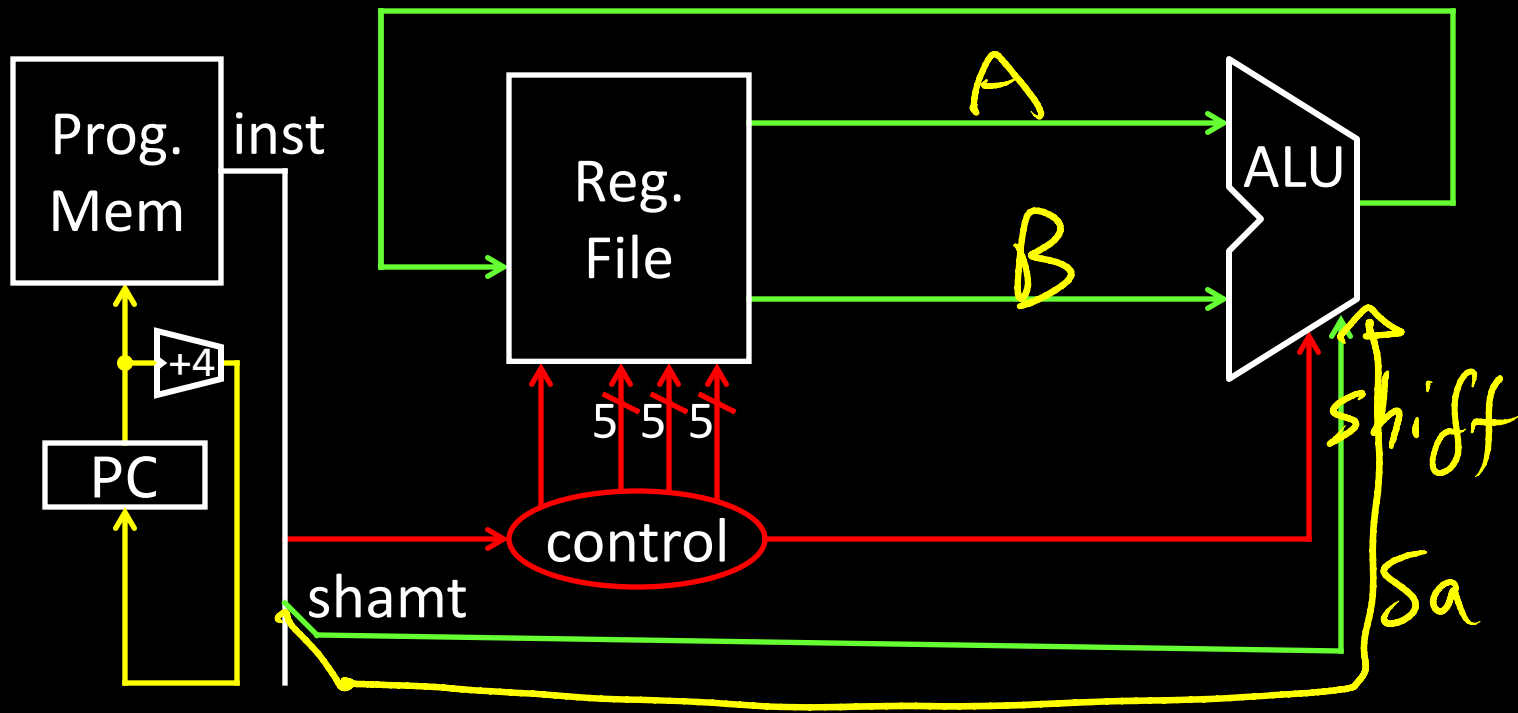
= Babbage's engine + speed + reliability – hand crank



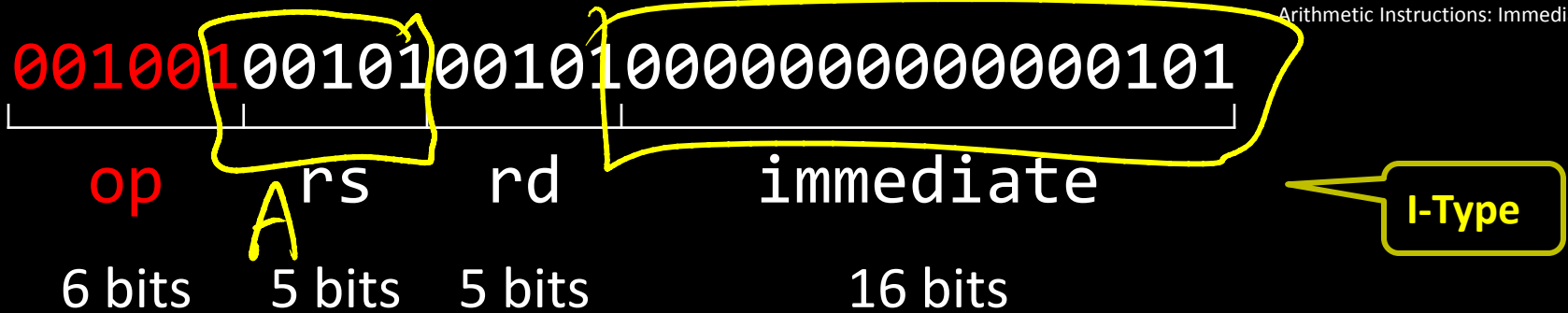
R-Type

op	func	mnemonic	description
0x0	0x0	SLL rd, rs, <u>shamt</u>	$R[rd] = R[rt] \ll \text{shamt}$
0x0	0x2	SRL rd, rs, shamt	$R[rd] = R[rt] \ggg \text{shamt}$ (zero ext.)
0x0	0x3	SRA rd, rs, shamt	$R[rd] = R[rs] \gg \text{shamt}$ (sign ext.)

ex: $r5 = r3 * 8$



$$r3 = r1 * r5$$

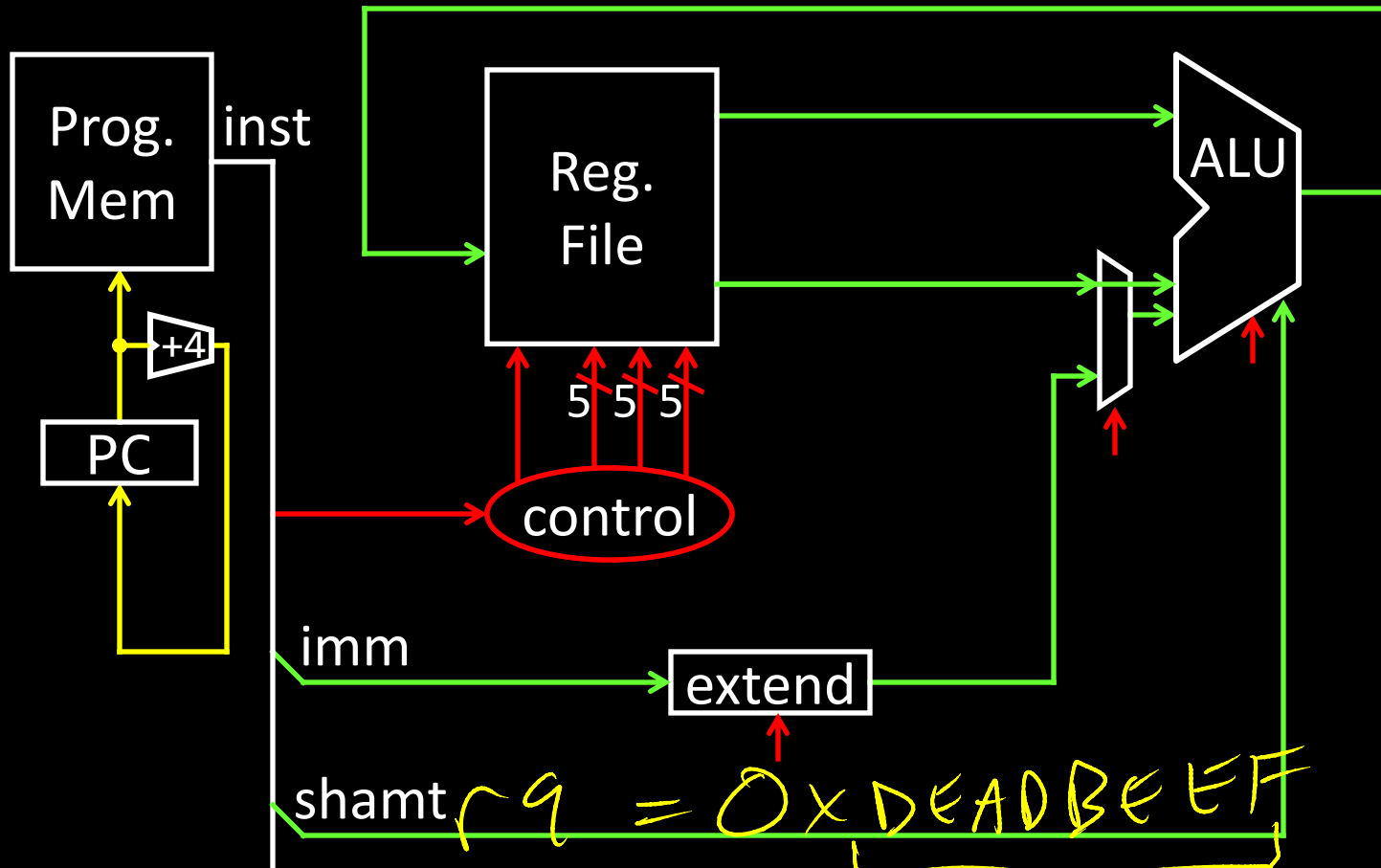


op	mnemonic	description
0x9	ADDIU rd, rs, imm	$R[rd] = R[rs] + \text{sign_extend}(\text{imm})$
0xc	ANDI rd, rs, imm	$R[rd] = R[rs] \& \text{zero_extend}(\text{imm})$
0xd	ORI rd, rs, imm	$R[rd] = R[rs] \mid \text{zero_extend}(\text{imm})$

ex: r5 += 5

ex: r9 = -1

ex: r9 = 65535



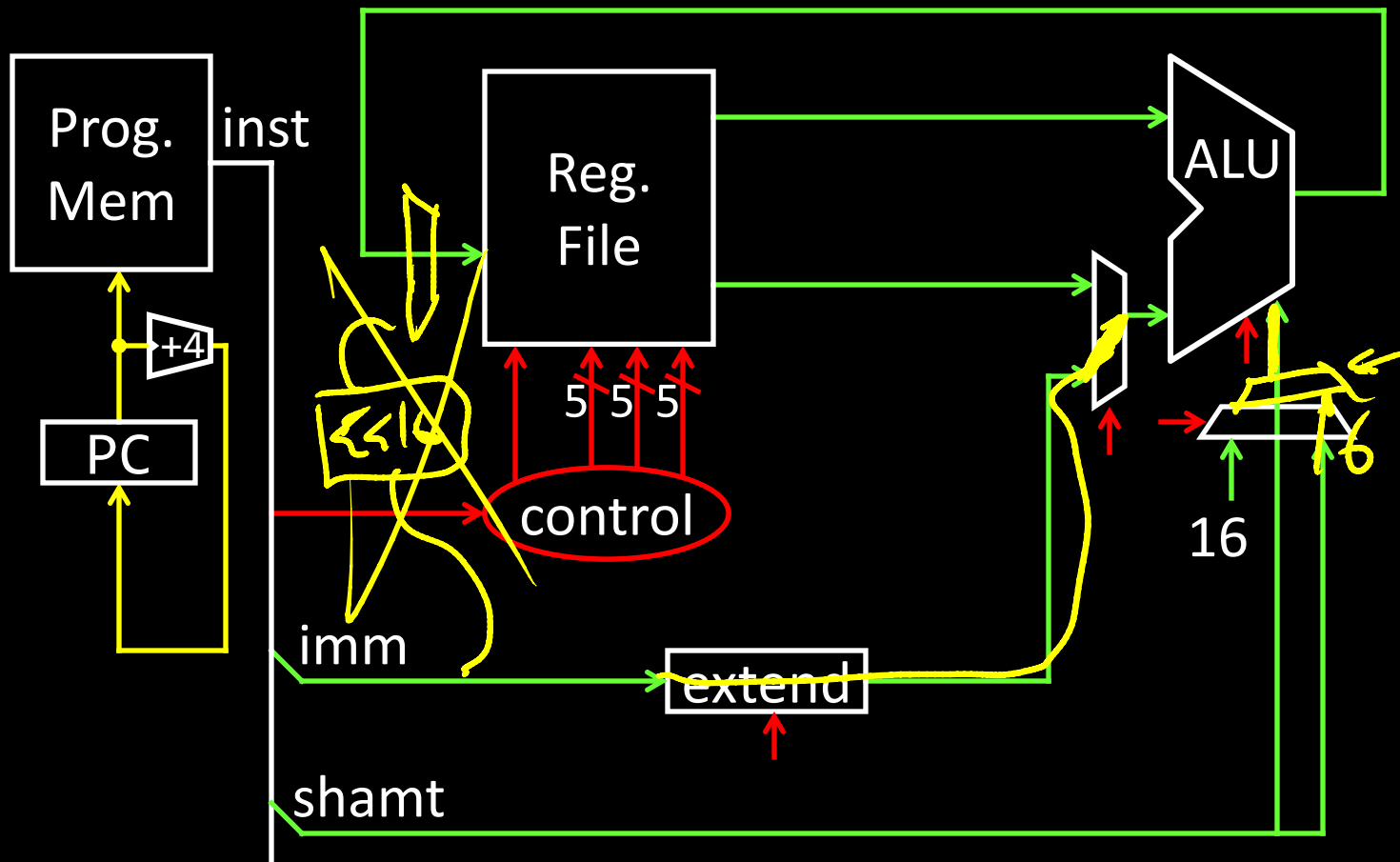
00111100000001010000000000000101

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

I-Type

op	mnemonic	description
0xF	LUI rd, imm	$R[rd] = \text{imm} \ll 16$

ex: r5 = 0xdeadbeef
 (load) 32 bits imm.



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

101001001010000100000000000000010

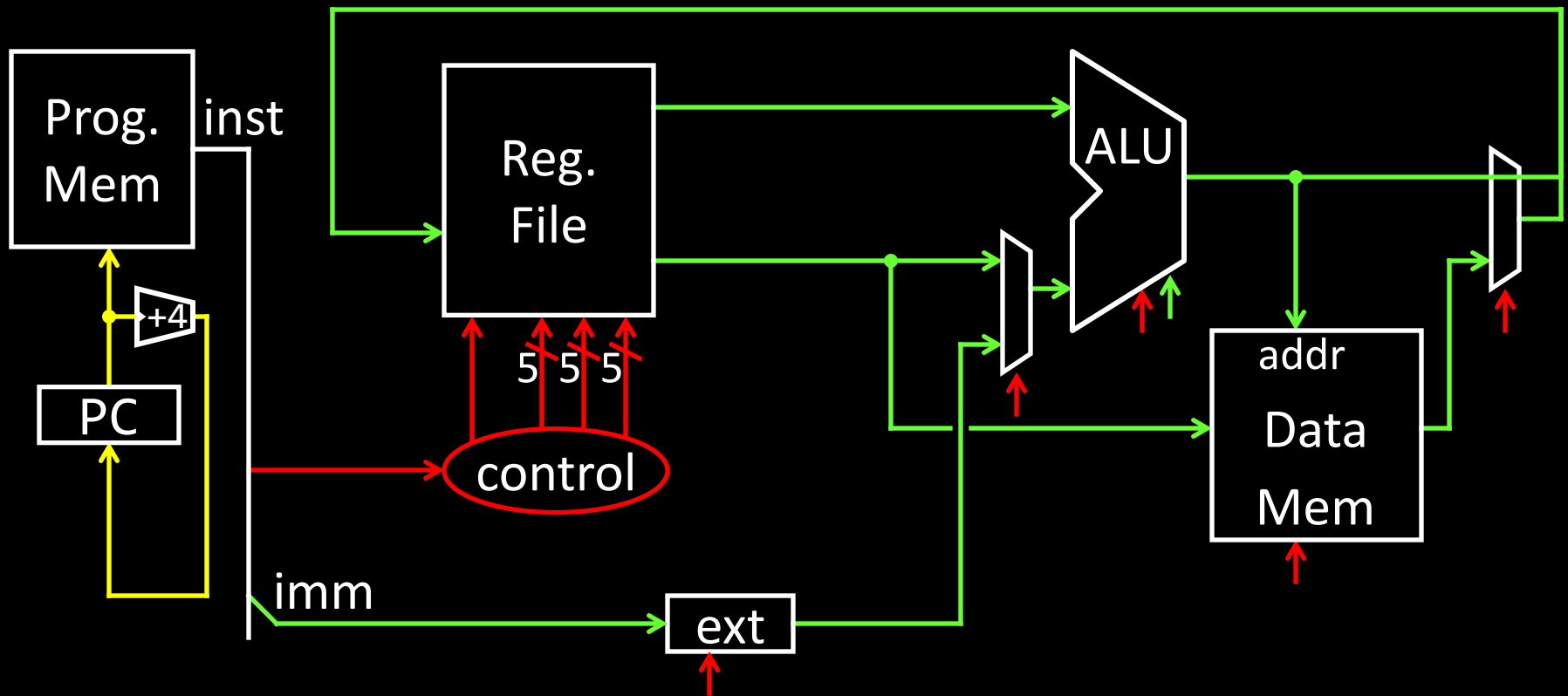
op rs rd offset
 6 bits 5 bits 5 bits 16 bits

I-Type

base + offset
addressing

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

signed
offsets



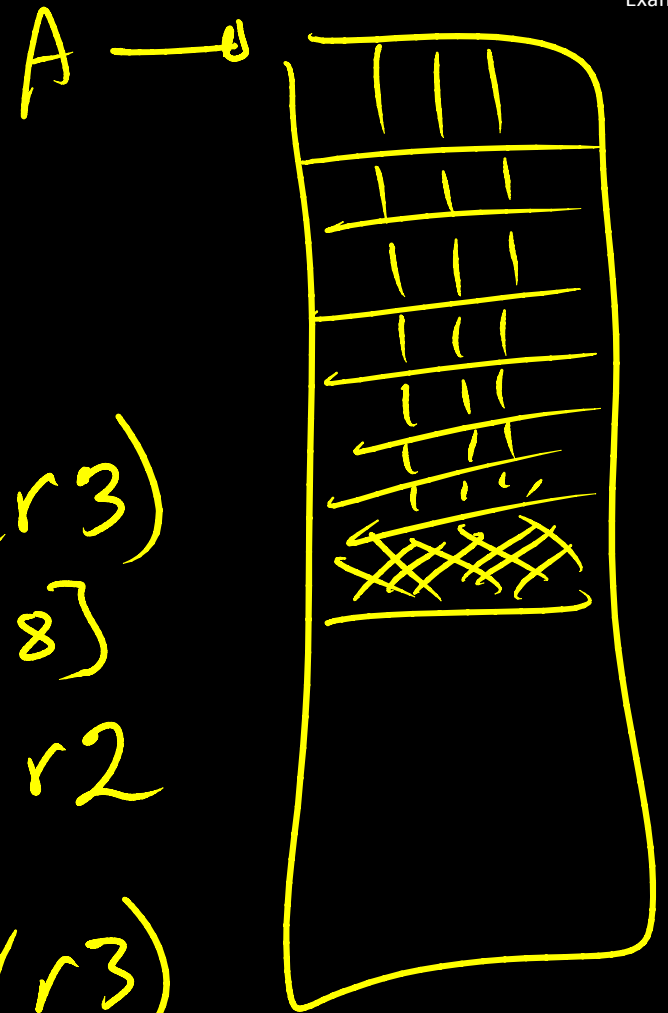
$r2$ $r3$
 $\downarrow$ $\downarrow$
int h, A[];
 A[12] = h + A[8];

LW $r4, 32(r3)$

$r4 = A[8]$

ADD $r5, r4, r2$

SW $r5, 48(r3)$



Examples:

5

r5 contains ~~0xDEADBEEF~~

sb r5, 2(r0)

lb r6, 2(r0)

sw r5, 8(r0)

lb r7, 8(r0)

lb r8, 11(r0)



	0x00000000
	0x00000001
	0x00000002
	0x00000003
	0x00000004
	0x00000005
	0x00000006
	0x00000007
5 0	0x00000008
0 0	0x00000009
0 0	0x0000000a
0 5	0x0000000b
	...
	0xffffffff

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	12 34		56 78	
as 1 word	0x12345678			

00001010100001001000011000000011

op

6 bits

immediate

26 bits

J-Type

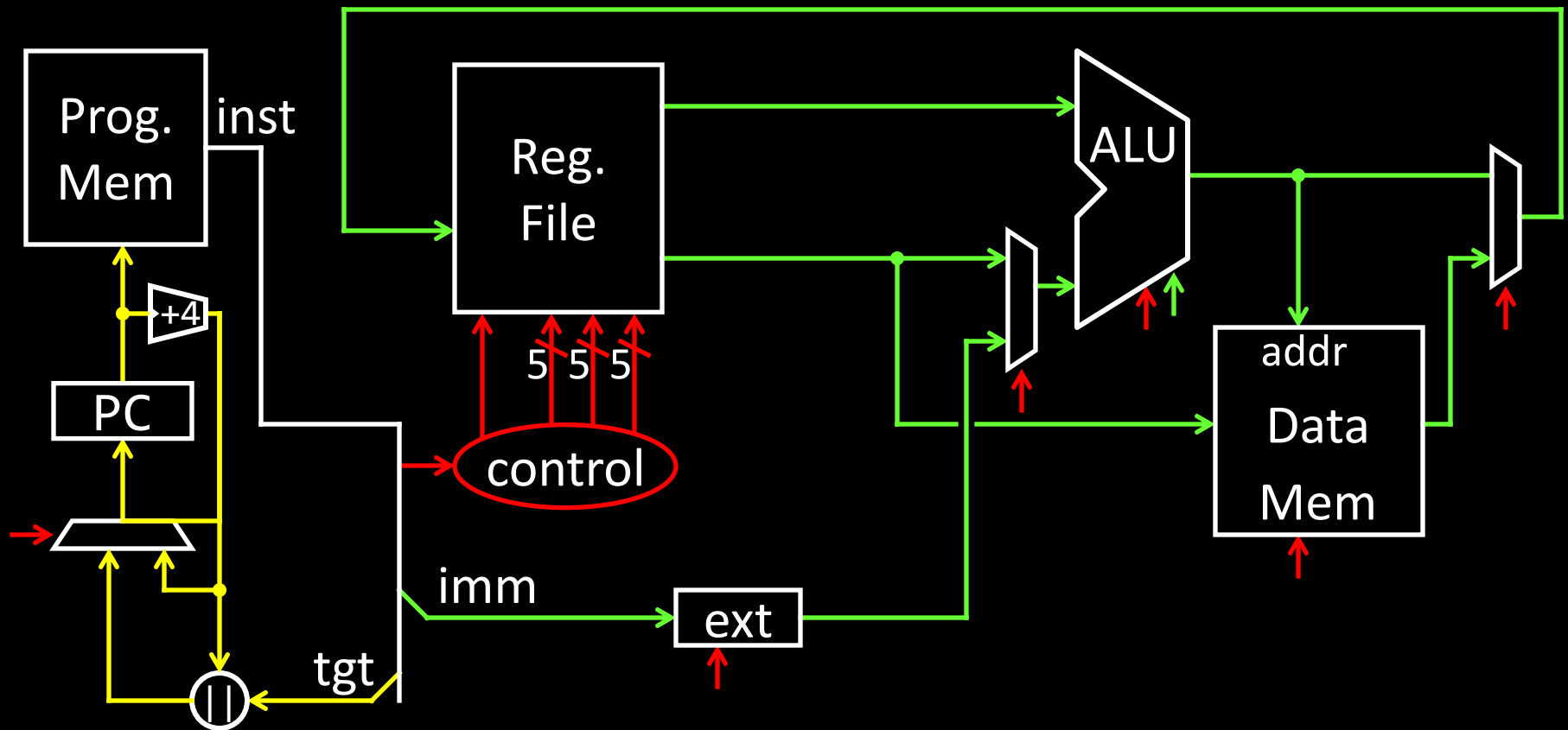
op	mnemonic	description
0x2	J target	$PC = (PC+4)_{32..29} \parallel \text{target} \parallel 00$

Absolute addressing for jumps

- Jump from 0x30000000 to 0x20000000? **NO** Reverse? **NO**
 - But: Jumps from 0x2FFFFFFF to 0x3xxxxxxx **are** possible, but not reverse
- Trade-off: out-of-region jumps vs. 32-bit instruction encoding

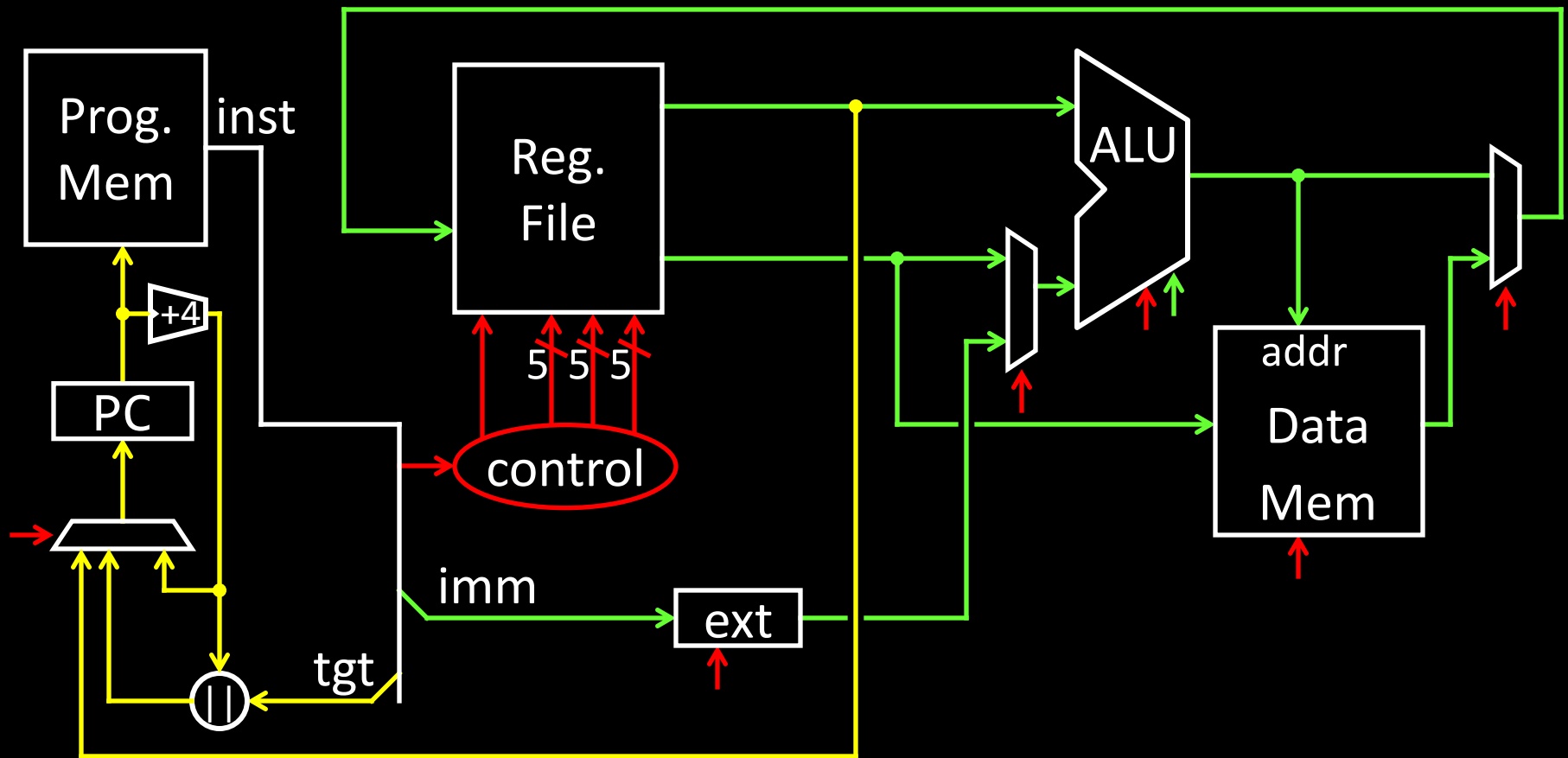
MIPS Quirk:

- jump targets computed using *already incremented* PC





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



jump to 0xabcd12345

~~J 0xabcd1234~~

```

LUI    r3, 0xabcd
ORI    r3, r3, 0xf2345
JR     r3

```

assume $0 \leq r3 \leq 1$

if ($r3 == 0$) jump to 0xdecafe0 $= X$

else jump to 0xabcd1234 $= Y$

$$r5 = Y - X$$

$$r5 = X + r3 * r5$$

$$\text{OR } r5$$

assume $0 \leq r3 \leq 1$

if ($r3 == 0$) jump to 0xdecafe0

else jump to 0xabcd1234

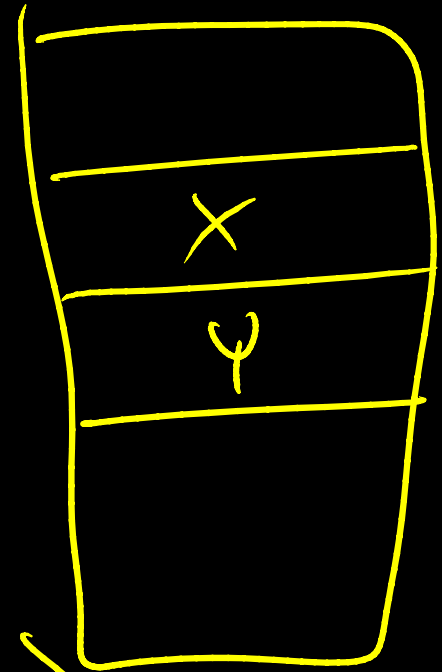
$= \begin{matrix} X \\ 0 \end{matrix}$

$r4 = X$

sw $r4$, 0($r0$)

$r4 = 4$

sw $r4$, 4($r0$)



0001000010100001000000000000000011

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 $PC = PC+4 + (\text{offset} \ll 2)$
0x5	BNE rs, rd, offset	if $R[rs] \neq R[rd]$ then $PC = PC+4 + (\text{offset} \ll 2)$

^{r5} if (i == ^{r6} j) { i = i * 4; } - if
 else { j = i - j; } - else

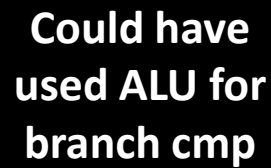
BNE r5, r6, ELSE (+8) (~~+4~~)

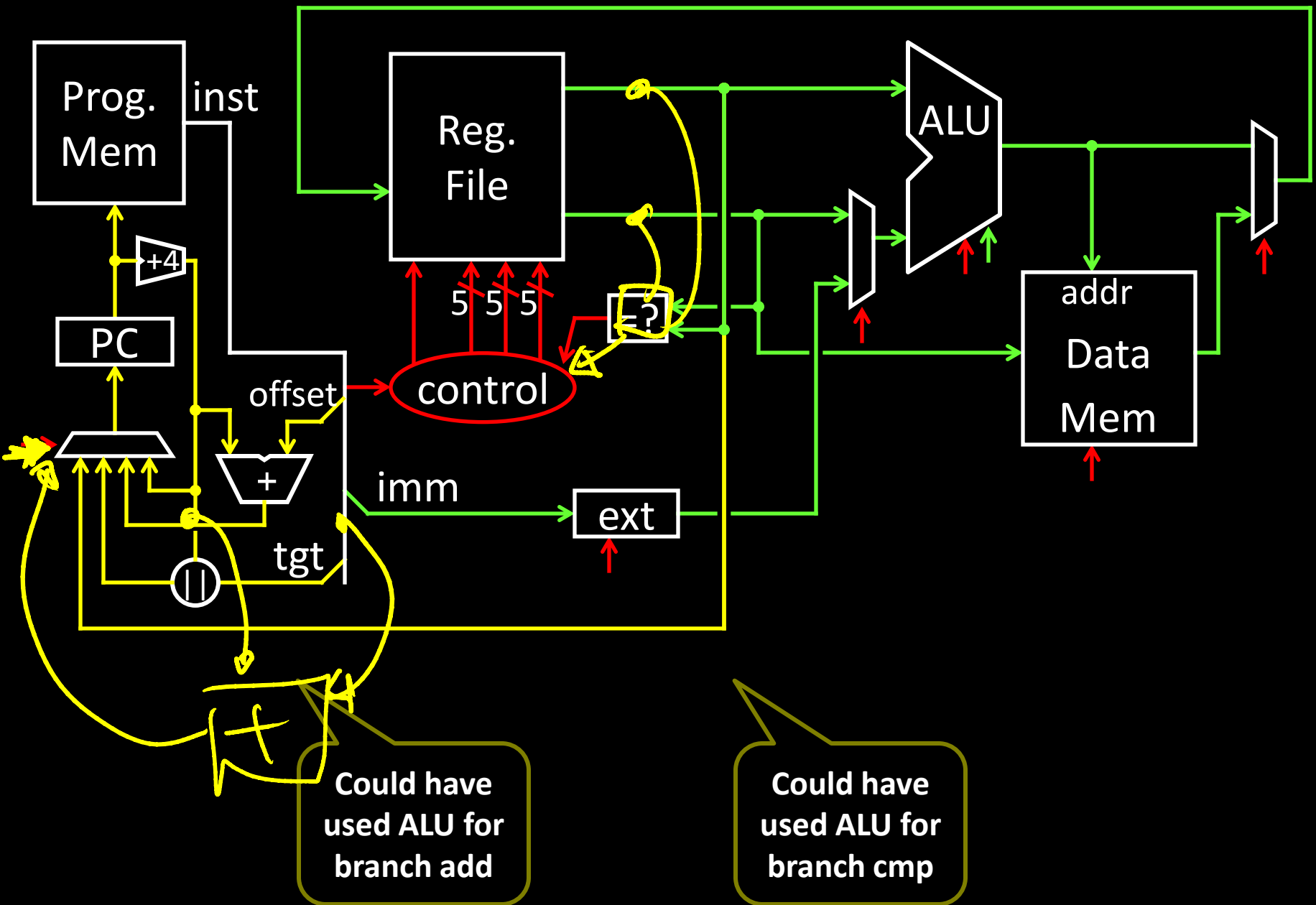
SLL r5, r5, 2

BEQ r0, r0, BOT (+4) (~~+8~~)

ELSE: SUB r6, r5, r6

BOT:





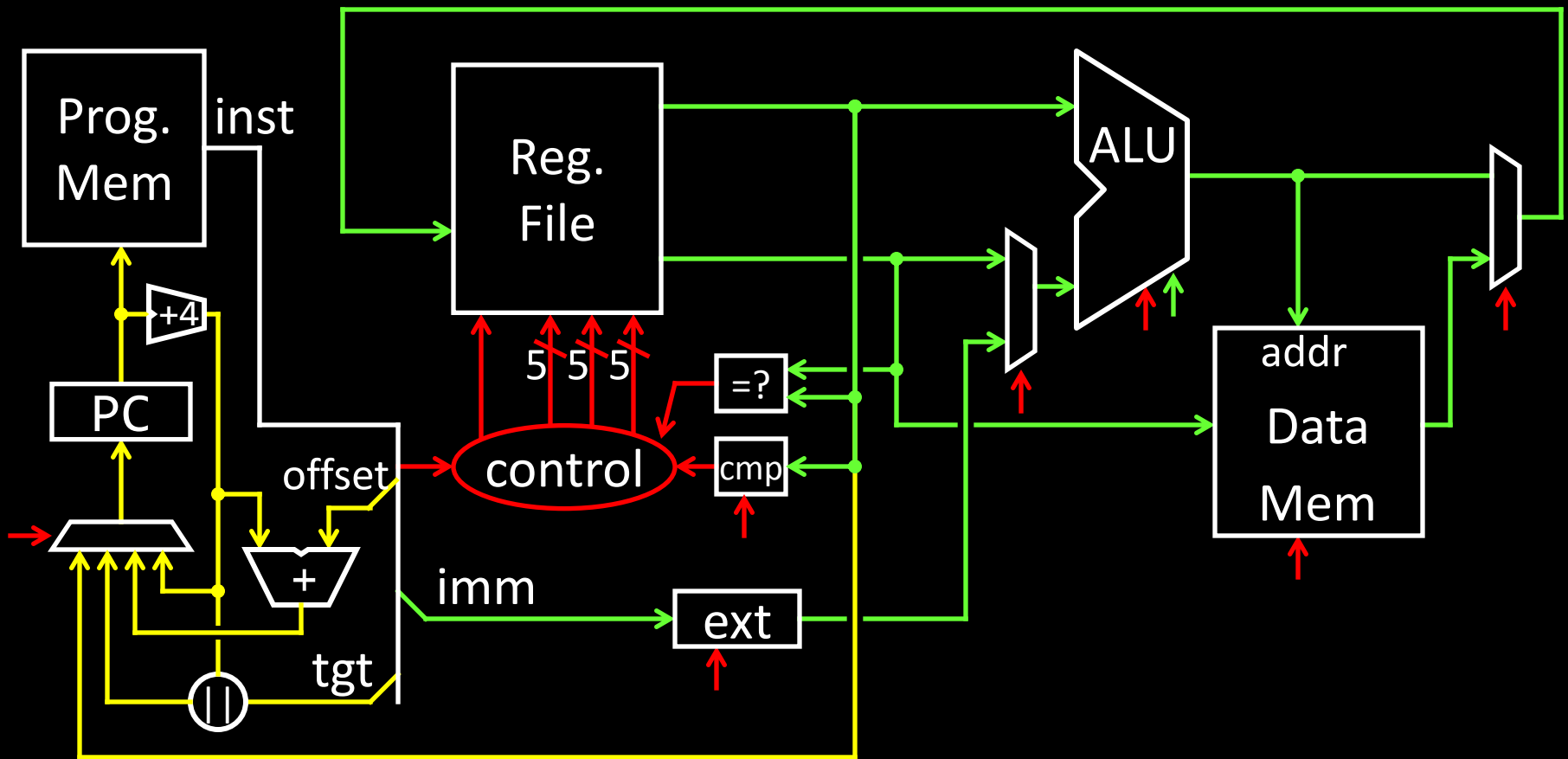
000001001010000100000000000000010

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 $PC = PC + 4 + (\text{offset} \ll 2)$
0x1	0x1	BGEZ rs, offset	if $R[rs] \geq 0$ then $PC = PC + 4 + (\text{offset} \ll 2)$
0x6	0x0	BLEZ rs, offset	if $R[rs] \leq 0$ then $PC = PC + 4 + (\text{offset} \ll 2)$
0x7	0x0	BGTZ rs, offset	if $R[rs] > 0$ then $PC = PC + 4 + (\text{offset} \ll 2)$



Could have
used ALU for
branch cmp

00001100000001001000011000000010

op

immediate

6 bits

26 bits

J-Type

op	mnemonic	description
0x3	JAL target	$r31 = PC + 8$ $PC = (PC + 4)_{32..29} \text{target} 00$

