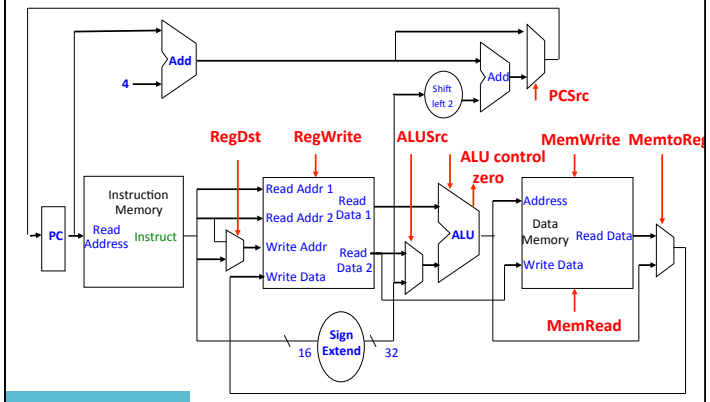
Control

A simple implementation

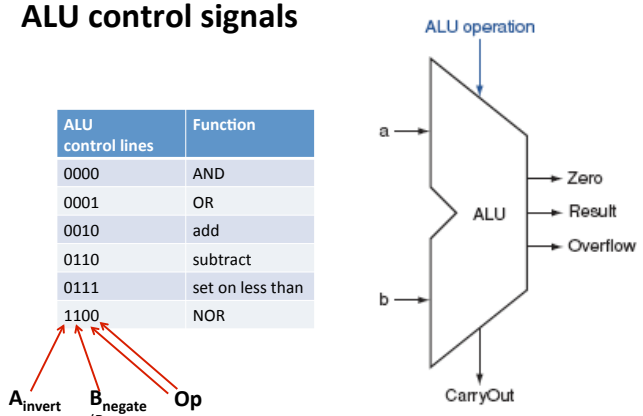
We built the instrument. Now we read music and play it...



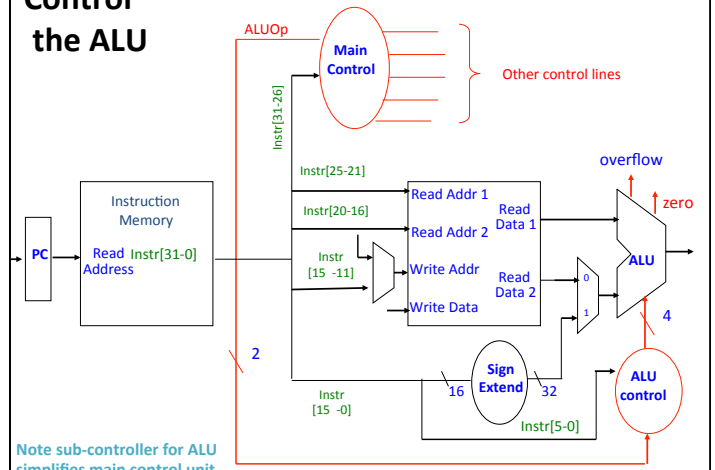
Next: control

ALU control signals

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR



Control the ALU



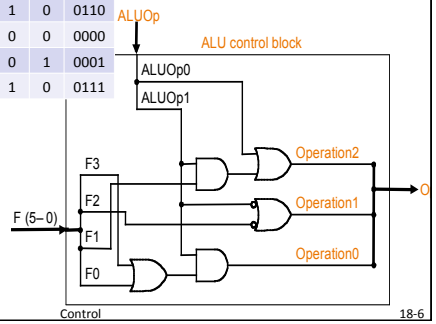
Note sub-controller for ALU simplifies main control unit.

Using ALUOp and Funct field to set ALU control bits

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	and	0000
R-type	10	OR	100101	or	0001
R-type	10	set on less than	101010	set on less than	0111

From truth table to ALU control

ALUOP		Funct field						Op
ALUOp 1	ALUOp 0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
0	1	X	X	X	X	X	X	0110
1	0	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	0	X	X	0	1	0	0	0000
1	0	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

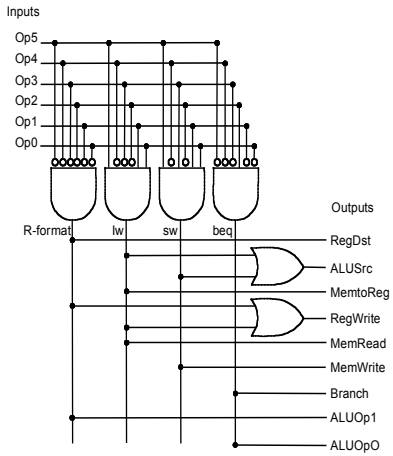


Main control unit

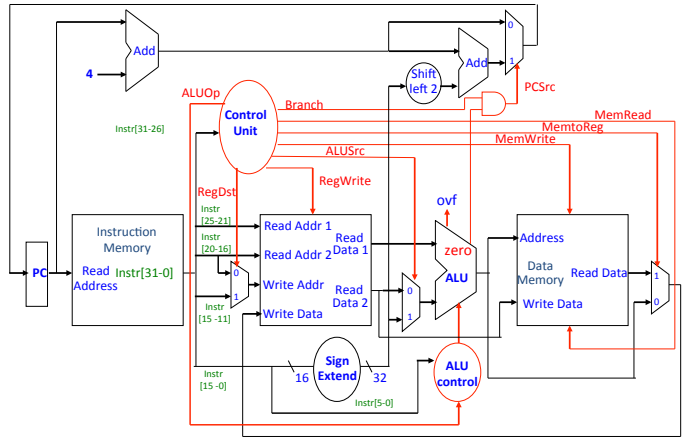
input; 6-bit opcode

output: 9 control bits

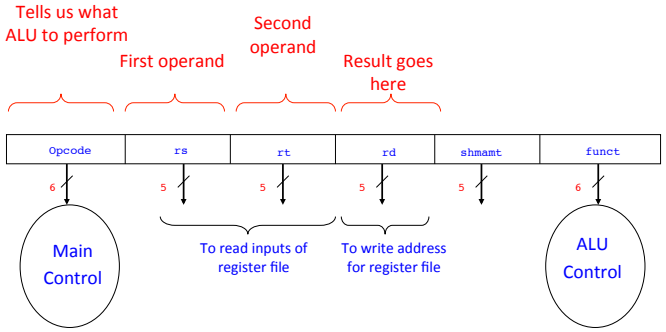
Opcode	Instruct
000000	R-format
100011	lw
101011	sw
000100	beq



The data path with control signals

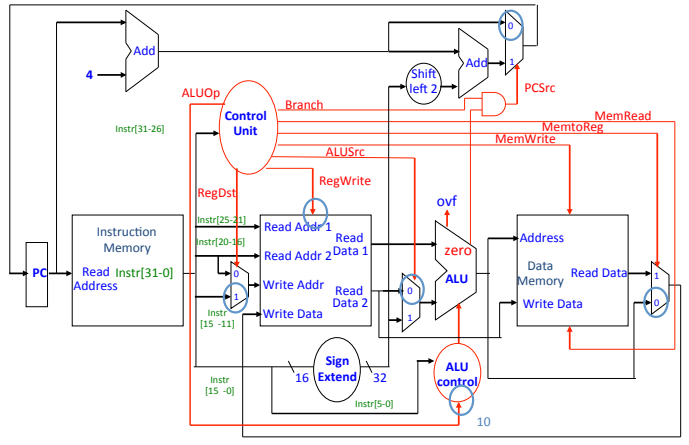


We start with R-instructions*

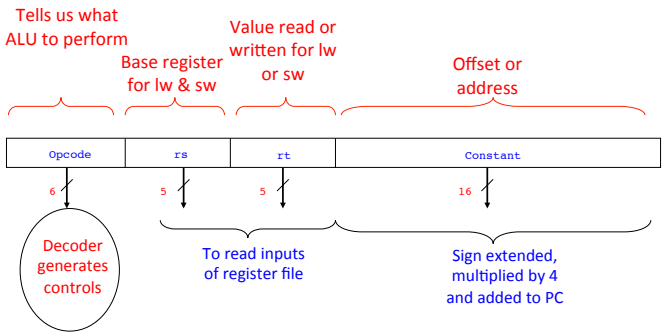


*R-type (for register) or R-format include `add`, `sub`, `and`, `or`, and `slt`.

R-instructions data and control signals

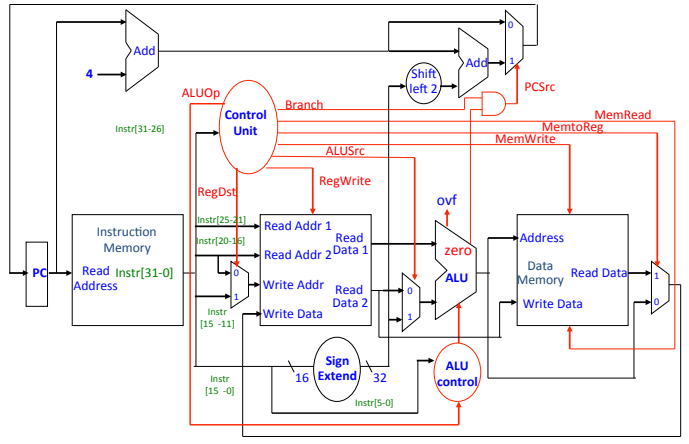


I-instructions come next

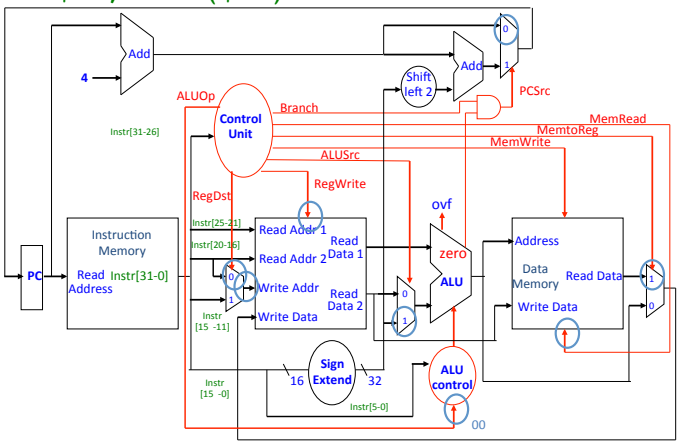


*I-type include `lw`, `sw`, `beq`, and `bnq`. Recall `lw $t0, 1200($t1)`.

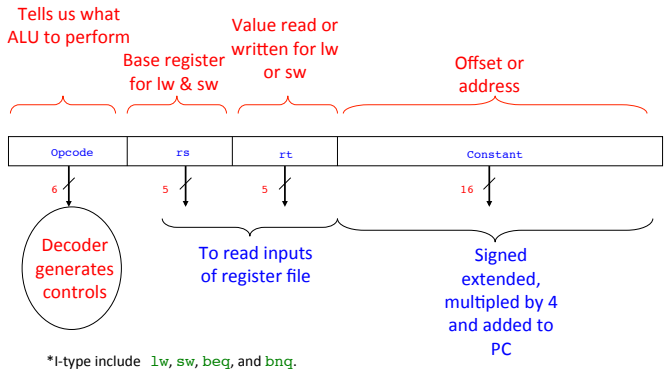
I-instructions data and control signals



```
lw $t0, 1200($t1)
```

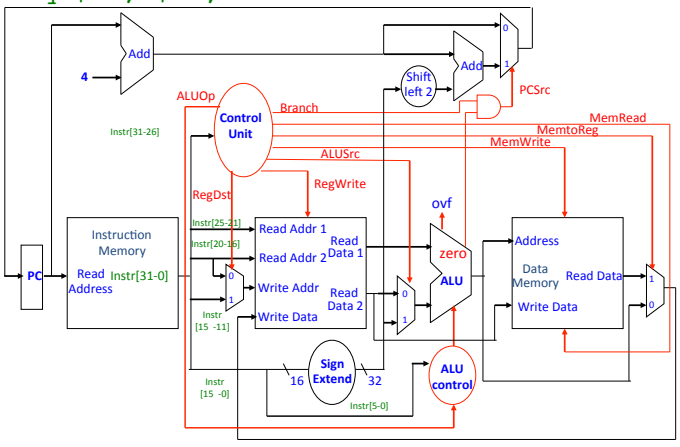


Branch instructions are also I-format

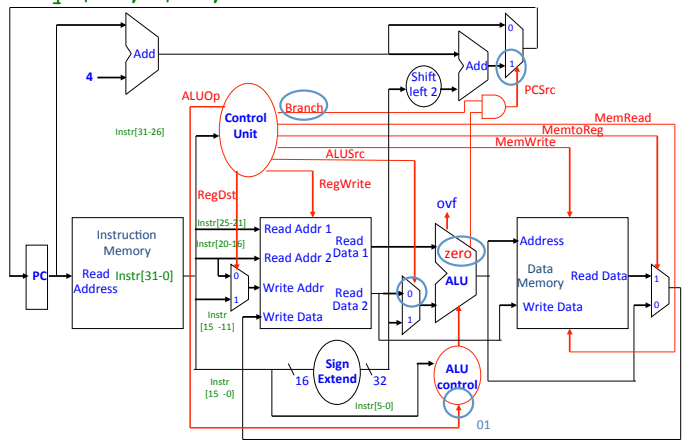


*I-type include `lw`, `sw`, `beq`, and `bnq`.


```
beq $s1, $s2, 100
```




```
beq $s1, $s2, 100
```

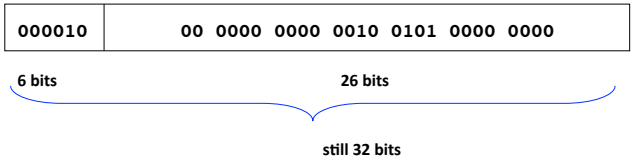


Control lines in sum

Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
`lw`	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0
`beq`	X	0	X	0	0	0	1	0	1

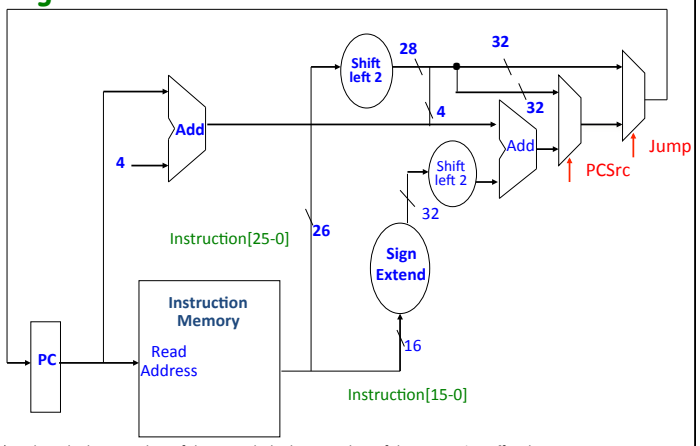
What about j?

`j 1000`



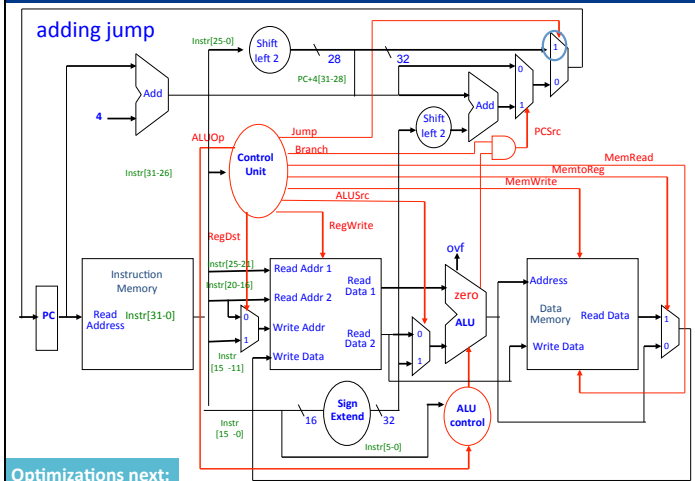
*Replace the lower 28 bits of the PC with the lower 26 bits of the instruction left shifted by 2 (multiplied by 4).

`j 1000`



*Replace the lower 28 bits of the PC with the lower 26 bits of the instruction offset by 2.

adding jump



Optimizations next: pipelining, caches

Single cycle pros and cons

Con Inefficient: clock must be timed to accommodate the slowest instruction.

Con May be wasteful of area since some functional units (e.g., adders) must be duplicated since they can not be shared during a clock cycle

Big Pro Simple and easy to understand

Optimizations next: pipelining, caches