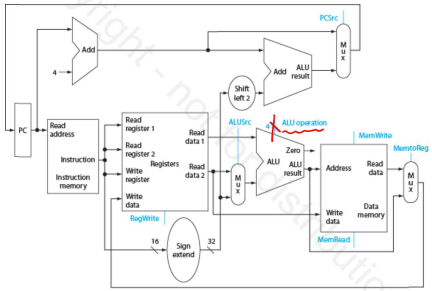
Single Cycle Datapath

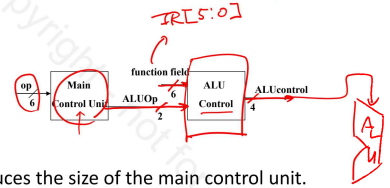


ALU Control

- Remember ALU Design:
  - Uses 4 control lines, 6 instructions implemented
- ALU is used for:
  - For load and store word instructions, memory address calculation by addition
  - For Branch instruction, subtraction
  - For R-type instructions, 5 actions: (and, or, add, subtract , set on less than) are performed depending on the 6-bit function field in the low order bits of the instruction

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

Multiple Levels of Decoding



- Reduces the size of the main control unit.
- Increases the speed of the control unit
- Important since control unit is performance critical

ALU Control

- Assume 2-bit ALUOp derived from opcode
  - Combinational logic derives ALU control

opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

Mapping ALU Operation Bits

- How-to implement the mapping 2-bit ALUOp field and the 6-bit funct field into three ALU operation fields:
  - Requires a small piece of logic that generates control signals
  - Create a truth table for particular combinations of function field and ALUOp bits.

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

Truth Table to Gates

ALUOp		Function code fields					
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0
X	1	X	X	X	X	X	X
1	X	X	X	X	X	1	X

a. The truth table for Operation2 = 1 (this table corresponds to the left bit of the Operation field in Figure C.2.1)

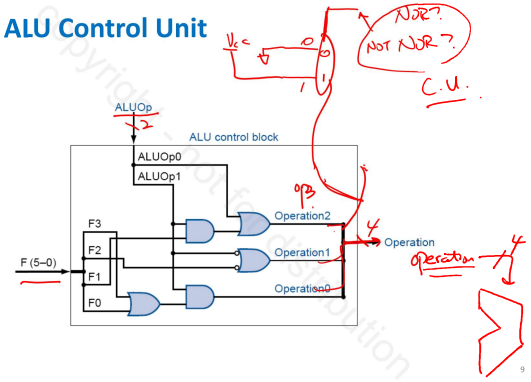
ALUOp		Function code fields					
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0
0	X	X	X	X	X	X	X
X	X	X	X	X	0	X	X

b. The truth table for Operation1 = 1

ALUOp		Function code fields					
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0
1	X	X	X	X	X	X	1
1	X	X	X	1	X	X	X

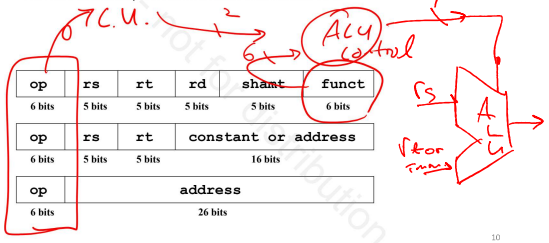
c. The truth table for Operation0 = 1

ALU Control Unit



Main Control Unit

- ALU control logic is part of the main control.
- Main control unit design:
  - We have to identify instruction fields and control lines

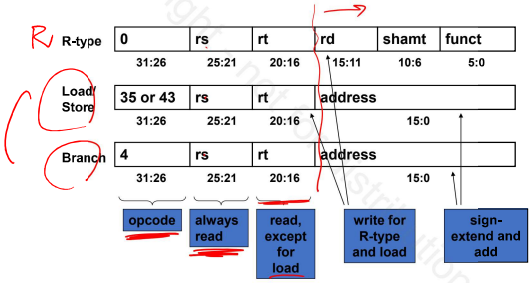


Observations on Instruction Formats

- Op field or opcode is always in bit locations 31:26. Op[5:0]
- Two registers to be read are **rs** and **rt** fields at positions 25:21 and 20:16. This is true for R-type, branch-equal; and for store
- The base register for load and store instructions is always in bit positions 25:21 (**rs**)
- The 16-bit offset for beq, load and store is always in positions 15:0
- The destination register is in one of two places. For load , it is in bit positions 20:16(**rt**), while for R-type instruction it is in bit positions 15:11 (**rd**)

Main Control Unit

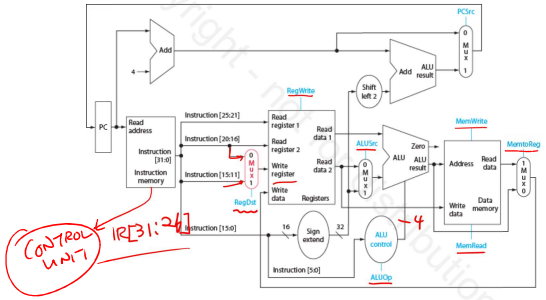
- Control signals derived from instruction



Datapath Modification

- A new multiplexor is required for selecting which field if the instruction is used to indicate the register number to be written
- ALU control block which generates the operation bits (ALU control signals)
- Seven single bit control lines and 2-bit ALUOp controls

MIPS Datapath



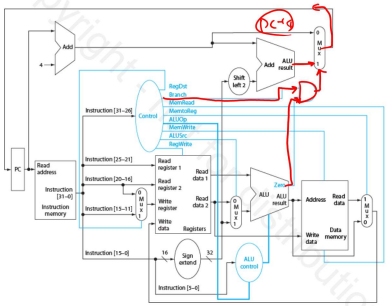
The Effect of Control Signals

Signal Name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16)	The register destination number for the Write register comes from the rd field (bits 15:11)
RegWrite	None	The register on the Write register input is written with the value on the Write data input
ALUSrc	The second ALU operand comes from the second register file output	The second ALU operand is the sign extended, lower 16 bits of the instruction
PCSRc	The PC is replaced by the output of the adder that computes the value of PC+4	The PC is replaced by the output of the adder that computes the branch target
MemRead	None.	Data memory contents designated by the address input are put on the Read data output
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input
MemtoReg	The value fed to the register Write data input comes from ALU	The value fed to the register Write data input comes from data memory

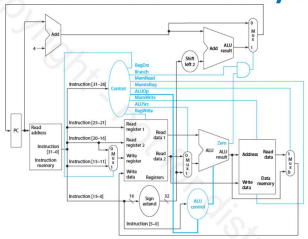
Branch Control Signal

- The main control unit can set all but one of the control signals based solely on the opcode field of the instruction.
  - PCSrc control line is the exception
  - It is set if the instruction is branch-on-equal and the Zero output of the ALU is true.
  - Use an AND gate to combine a **`Branch`** signal from the control unit and Zero signal out of the ALU
- Nine control signals can now be set on the basis of 6 input signals to the control unit which are the opcode bits

Simple Datapath with Control



Control Lines Determined by Opcode



Instruction	RegDst	ALUSrc	Mem to Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp2
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

X: don't care

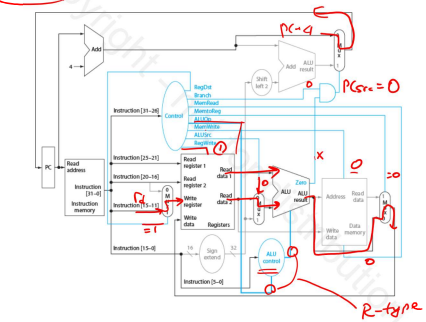
Operation of the Datapath

- How each instruction uses the datapath:
- R-type instruction example:

```
add $t1,$t2,$t3
```

- Four steps to execute the instruction:
  1. Instruction is fetched, PC is incremented
  2. Two registers, $t2 and $t3 are read and main control unit computes the setting of the control lines
  3. The ALU operates on the data read using function code to generate ALU function
  4. The result from the ALU is written into register file using bits 15:11 of instruction to select destination register $t1

R-type Datapath Operation



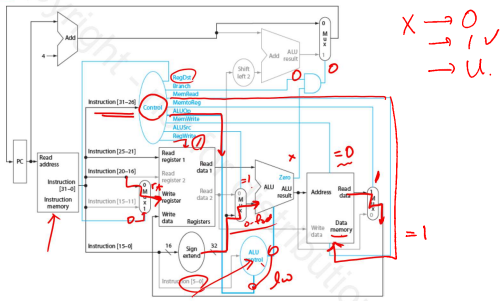
Load Instruction Datapath Operation

- Load word instruction execution:

```
lw $t1, offset($t2)
```

- Five steps for the execution:
  1. An instruction is fetched form instruction memory and PC is incremented
  2. A register ($t2) value is read from the register file
  3. The ALU computes the sum of the value read from the register file and the sign-extended, lower 16-bits of the instruction
  4. The sum from the ALU is used as the address for the data memory
  5. The data from the memory unit is written into register file; the register destination is given by bits 20:16 of the instruction ($t1)

Load Instruction Datapath Operation



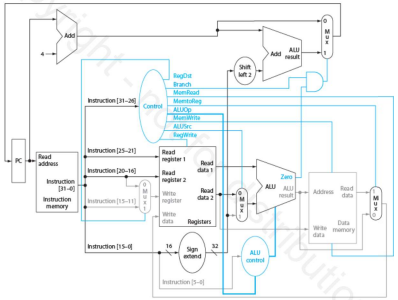
beq Instruction Datapath Operation

- Branch-on-equal instruction execution:

```
beq $t1,$t2,offset
```

- Four steps for the execution:
  1. An instruction is fetched from instruction memory and PC is incremented
  2. Two registers, $t1 and $t2 are read from the register file
  3. The ALU performs a subtract on the data values read from the register file. The value of PC+4 is added to the sign-extended, lower 16 bits of the instruction (offset) shifted left by two; the result is the branch target
  4. The Zero result from ALU is used to decide which adder result to store into PC

beq Instruction Datapath Operation



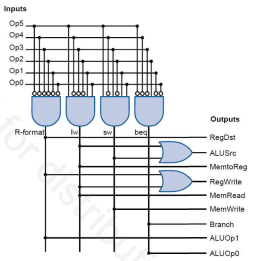
Finalizing the Control Unit

- Create a truth table for each of the outputs based on the binary encoding of the opcodes

Control	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Control Function Implementation

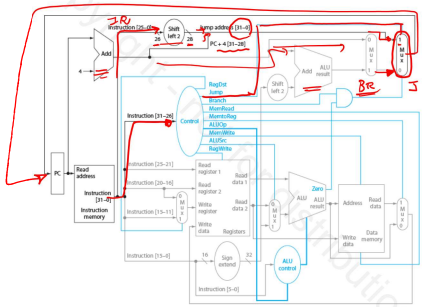
- 64 possible opcodes indicate large number of gates for control implementation
- Control Unit can be easily implemented from the truth table by PLA (programmable logic arrays)
- Two-level logic:
  - AND gate array followed by OR gate array



Implementing Jumps

- Extend datapath and control to include the jump instruction
- Similar to branch instruction; however unconditional.
- Lower 2 bits of a jump address are always 00_{two}. The next lower 26 bits come from immediate field in the instruction. The upper 4 bits of the address come from the PC of jump instruction + 4. Thus a jump is implemented by storing into PC the concatenation of:
  - The upper 4 bits of the current PC+4
  - The 26 bit immediate field of jump instruction
  - The bits 00_{two}
- An additional multiplexor is required to select the source for the new PC value, which is either incremented PC, branch target PC or *jump* target PC.
- On additional control signal called `jump` is asserted when the instruction is jump, when the opcode is 2.

Implementing Jumps



Summary

- Single cycle is inefficient both in performance and hardware cost:
  - Clock cycle is equal to the worst case delay for all instructions
  - Reducing the delay of common case does not improve worst case cycle time
  - Violates the design principle:
    - Making the common case fast
  - Each functional unit can be used only once; some functional units are duplicated
- Solution:
  - Use shorter clock cycles and multiple clock cycles for each instruction
  - Use Pipelining: overlapping the execution of multiple instructions increasing hardware utilization and performance