Concepts Introduced

- single cycle datapath
- single cycle control
- multicycle datapath
- multicycle control

Processor Implementation

- The implementation of a processor consists of two components, the *datapath* and the *control*.
- datapath
  - Elements that hold data, including the program counter, register file, processor instruction memory, and processor data memory.
  - Elements that operate on data, including the ALU and adders.
  - The buses that allow data to be sent from one element to another.
- The control is the component of the processor that commands the datapath when and how to route and operate on data.
- types of implementation
  - single cycle
  - multicycle
  - pipelined

Implementing a Subset of MIPS Instruction Set

- We will go over an implementation that includes the following subset of the MIPS instruction set.
  - memory reference instructions: *load word* (`lw`) and *store word* (`sw`).
  - arithmetic/logical instructions: *integer addition* (`add`), *integer subtraction* (`sub`), *bitwise and* (`and`), *bitwise or* (`or`), and *set less than* (`slt`).
  - transfer of control instructions: *branch equal* (`beq`) and *jump* (`j`).

General Instruction Operation

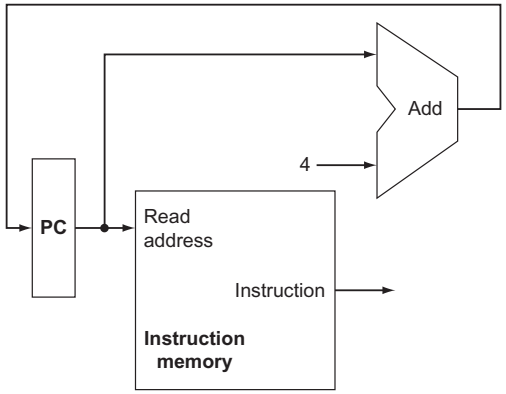
- Fetch the instruction at the address in the program counter (PC) from the instruction memory.
- Determine the fields within the instruction.
- Perform the operation indicated by the instruction.
- Update the PC so the next instruction can be fetched in the following cycle.

Common Instruction Steps

- For every type of instruction, the first few steps are identical.
  - Fetch the instruction at the address in the program counter (PC) from the instruction memory.
  - Determine the values associated with the instruction fields.
    - Read the register file using the appropriate fields of the instruction that contain the source register numbers.
    - Sign extend the immediate value within the instruction.
- After the first few steps, the actions will depend on the instruction type, but some actions will be common to different instructions.

Portion of Data Path for Instruction Fetch

- The figure below shows the datapath for fetching instructions and incrementing the PC.
- The PC is incremented by 4 since each instruction is four bytes.



R-Format Instructions

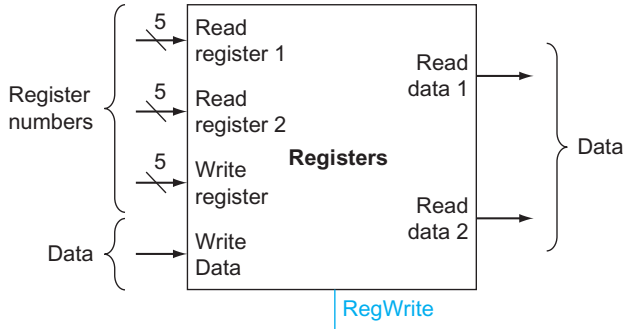
- The R-format instructions include the *add*, *sub*, *and*, *or*, and *slt* instructions.
- Each R-format instruction reads two register values from the register file specified by the *rs* and *rt* fields and writes the result to a register in the register file specified by the *rd* field.

Name	Fields					
Field Size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
R format	op	rs	rt	rd	shamt	funct

op – instruction opcode
rs – first register source operand
rt – second register source operand
rd – register destination operand
shamt – shift amount
funct – additional opcodes

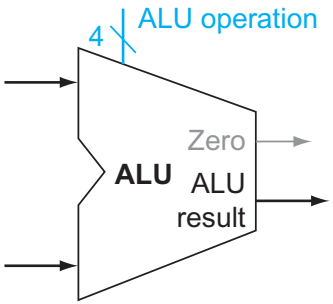
Register File

- The register file has two read ports and one write port.
- Each read port has one 5-bit input to specify which register to access and one 32-bit output containing the register value.
- The write port has one 5-bit input register number and a 32-bit input containing the value to be assigned to that register.
- There is also a signal indicating if the register file is to be updated.

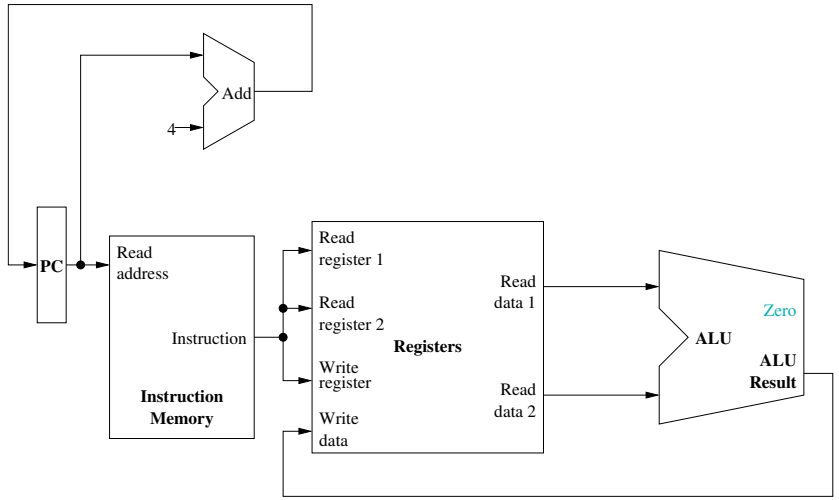


ALU

- The ALU has two 32-bit inputs and one 32-bit output, which is the *ALU result*.
- The *Zero* output is a control line used for branches.
- The *ALU operation* is a 4-bit control line used to indicate which ALU operation is to be performed.



Datapath for Only R-Type Instructions



I-Format Instructions

- The I-format instructions include the *load word* (`lw`), *store word* (`sw`), and *branch equal* (`beq`) instructions.
- Each I-format instruction accesses register values from the register file specified by the *rs* and *rt* fields, where the *rs* field is always used to read a register and the *rt* field is used to read or write a register, depending on the instruction type.
- Each I-format instruction either sign extends or zero extends the *immed* value to make it a 32-bit value. Branch instructions also left shift the *immed* field by 2 after sign extending the value.

Name	Fields			
Field Size	6 bits	5 bits	5 bits	16 bits
I format	op	rs	rt	immed

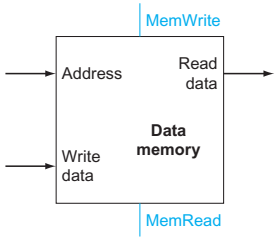
op – instruction opcode
rs – first register source operand
rt – second register source operand
immed – offsets/constants

Data Transfer Instructions

- The *lw* and *sw* instructions have the following assembly form.
  - `lw $`*rt*`,`*immed*`(`*$rs*`)`
  - `sw $`*rt*`,`*immed*`(`*$rs*`)`
- The memory address is computed by sign extending the 16-bit *immed* value to 32 bits and adding it the register value specified by the *rs* register number.
- The *rt* field for the *lw* instruction specifies the register that will be assigned the memory value and for the *sw* instruction the *rt* field specifies the register whose value will be stored in memory.

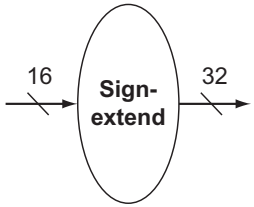
Data Memory Unit

- The data memory unit has four inputs and one output.
- The 32-bit *Address* input specifies the byte memory address where the data is to be accessed.
- The 32-bit *Write data* input specifies the data value to be written to memory by a *sw* instruction.
- The 32-bit *Read data* output contains the data value read from memory by a *lw* instruction.
- The *MemRead* and *MemWrite* signals indicate whether the data memory is to read or written, respectively.

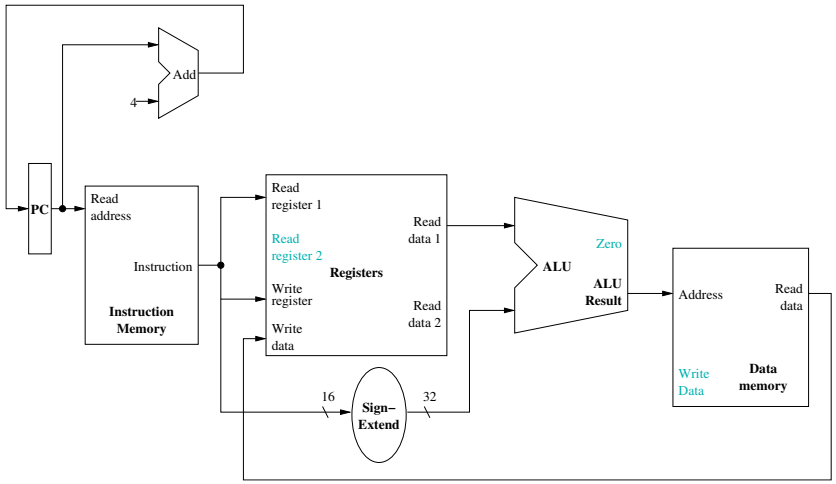


Sign Extension Unit

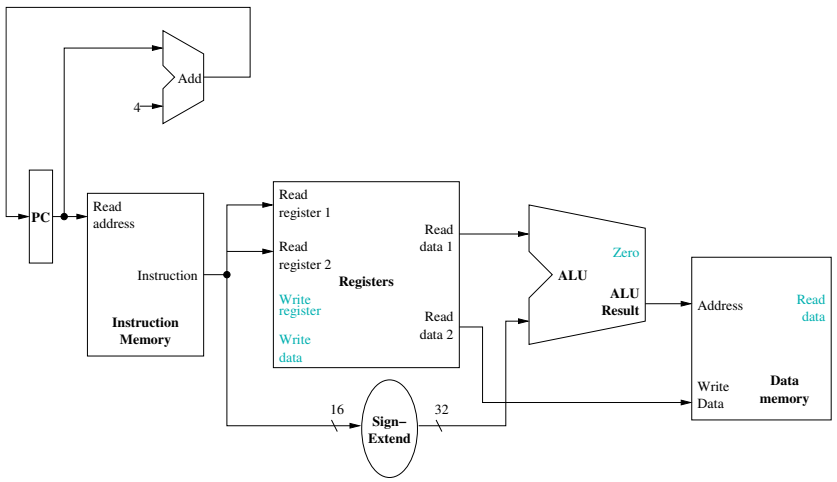
- The sign extension unit takes the 16-bit *immed* value as input and is sign extended to a 32-bit result.



Datapath for Only the `lw` Instruction



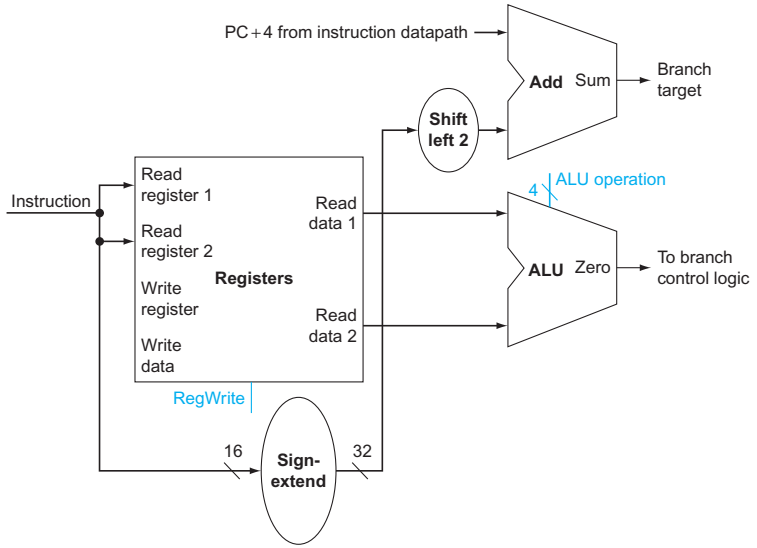
Datapath for Only the `sw` Instruction



Branch Instructions

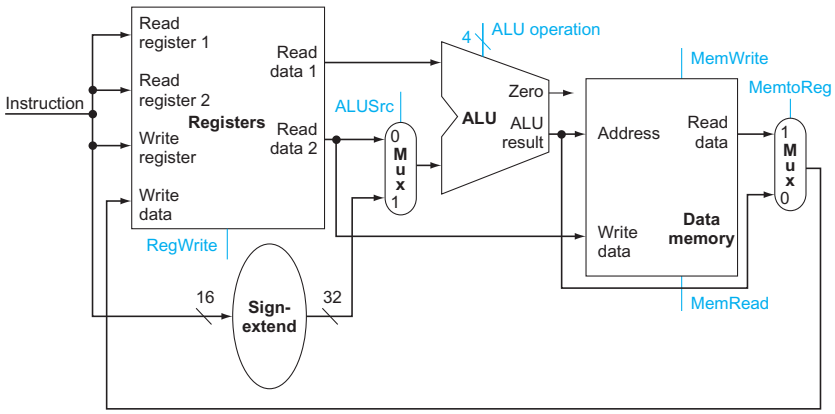
- The *beq* instruction also uses the I format.
- The *beq* instruction tests if the two values loaded from the *rs* and *rt* registers have the same value.
- If so, it then control is transferred to address that is the sum of address of the following instruction ($PC+4$) and the 16-bit *immed* value sign extended to 32 bits and left shifted by 2. A dedicated adder is used to calculate the target address.
- The ALU is specified to perform a subtraction operation and a *Zero* signal is output from the ALU to indicate if the two register values were equal.

Portion of Datapath for *beq* Instruction

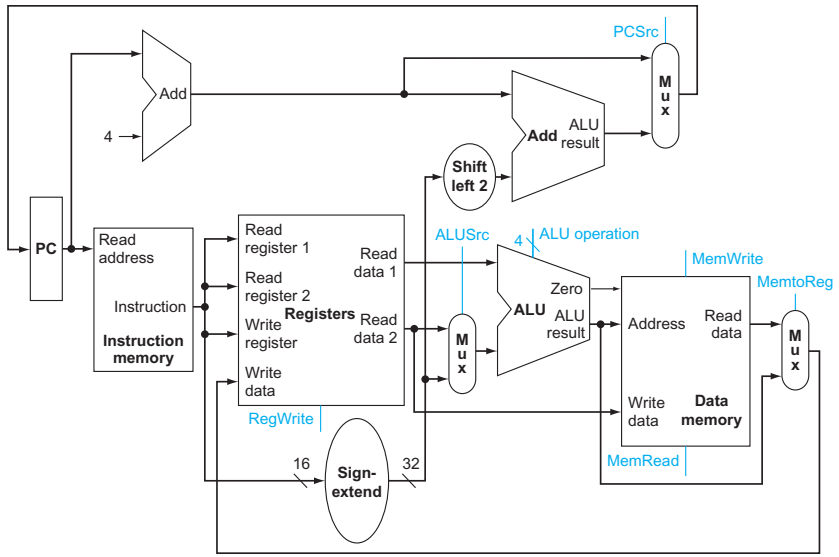


Datapath for Memory and Arithmetic/Logical R-Type Insts

- When a data element is shared between different instruction classes, a multiplexor is used when there are multiple connections to an input of the element.



Datapath for Memory, Arithmetic/Logical, and Branch Insts



J-Format Instructions

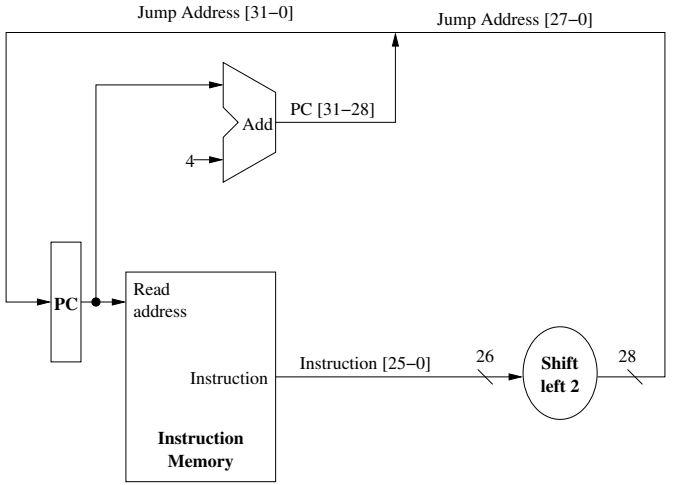
- The J-format instructions include the *jump* (j) instruction.
- The 26-bit *targaddr* field is left shifted by 2 and the result replaces the 28 least significant bits of the PC.

Name	Fields	
Field Size	6 bits	26 bits
J format	op	targaddr

op – instruction opcode targaddr – jump/call target

Datapath for *j* Instruction

- The figure below shows the datapath needed for a *j* instruction.



Control

- The *control* is the component of the processor that commands the datapath how to route and operate on data.
- Control is accomplished by sending the appropriate signals into the various data elements.
- The control for the MIPS includes signals to command the ALU, to select inputs for the various multiplexors, to indicate whether or not the register file is to be written, and to indicate whether or not the data memory is to be read or written.
- Most of these control signals are generated by inspecting the *opcode* and *funct* fields of the current instruction.

ALU Control Lines

- The MSB is the *Ainvert* line, the 2nd MSB is the *Binvert* line, and the two LSBs represent the *Operation* lines.

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

How the ALU Control Bits Are Set

- The *ALU control* bits are determined from the *opcode* and the *funct* fields.
- Assume *ALUOp* has been determined for each instruction.
- The XXXXXX value in the *Funct* field indicates that the value is not used and does not affect the result.

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

Bit Numbers for Fields of Instructions

- The following figure shows the three instruction formats along with the bit numbers of their fields.

Field	0	rs	rt	rd	shamt	funct
Bit positions	31:26	25:21	20:16	15:11	10:6	5:0

a. R-type instruction

Field	35 or 43	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

b. Load or store instruction

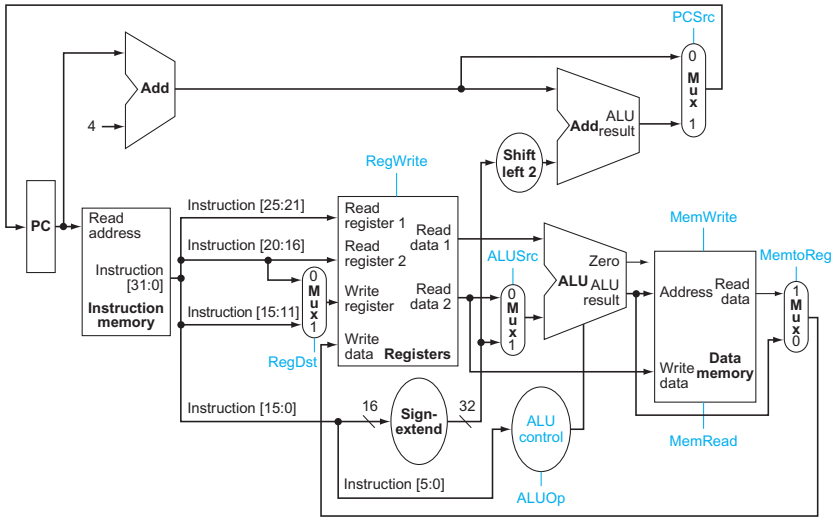
Field	4	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

c. Branch instruction

Use of Instruction Fields

- The *opcode* is contained in bits 31:26 and the *funct* is contained in bits 5:0. These two fields are used to determine the type of instruction.
- The two registers to be read are specified by the *rs* field in bits 25:21 and the *rt* field in bits 20:16. Both register values read are used in the *beq*, *sw*, and the R-type instructions.
- The base register for load and store instructions is specified by the *rs* field in bits 25:21.
- The 16-bit *address* field in bits 15:0 are used for the *beq*, *lw*, and *sw* instructions.
- The destination register for an *lw* is specified by the *rt* field in bits 20:16 or for an R-type instruction it is specified by the *rd* field in bits 15:11.

Datapath with All Control Lines Identified



Effects from Deasserting/Asserting 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 (Read data 2).	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 the ALU.	The value fed to the register Write data input comes from the data memory.

Datapath with the Control Signals

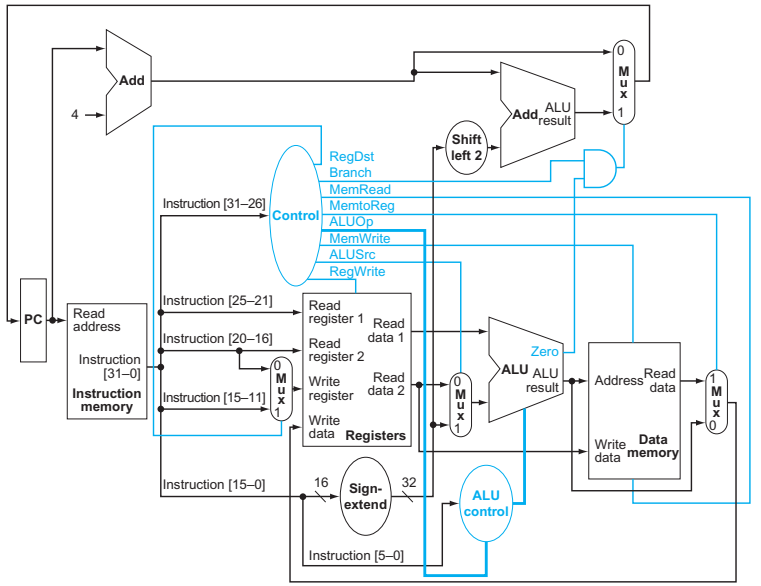
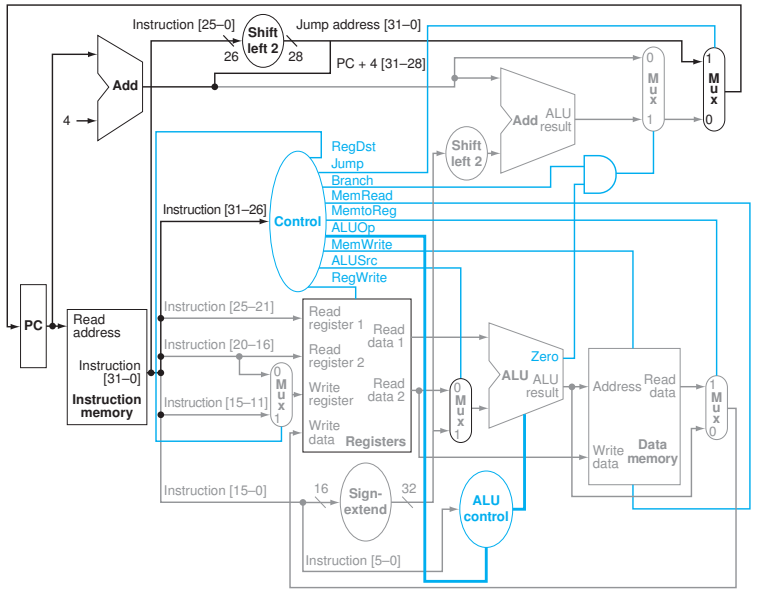


Table Showing Setting of Control Lines

- The setting of the nine control lines is determined by the *opcode* field.
- The X's indicate the value doesn't matter, which in this case is due to the register file not getting updated.
- The ALU control lines are also determined by the *funct* field.

Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
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

Control and Datapath Extended for a *j* Instruction



Cycle Time for the Single Cycle Implementation

- What is the longest path (slowest instruction) assuming 4ns for instruction and data memory, 3ns for ALU and adders, and 1ns for register reads or writes? Assume negligible delays for muxes, control unit, sign extend, PC access, shift left by 2, routing, etc.

Instruction Type	Instruction Memory	Register Read	ALU Operation	Data Memory	Register Write	Total
R-type	4	1	3		1	9
lw	4	1	3	4	1	13
sw	4	1	3	4		12
beq	4	1	3			8
j	4					4

Single Cycle Overview

- Single cycle implementation advantage is that it is simple to implement.
- disadvantages
  - The clock cycle will be determined by the longest possible path, which is not the most common instruction. This type of implementation violates the idea of making the common case fast.
  - May be wasteful with respect to area since some functional units, such as adders, must be duplicated since they cannot be shared during a single clock cycle.

Multicycle Implementation

- A multicycle implementation can avoid the disadvantages of a single cycle implementation with some additional complexity.
- We have gone over the following types of instructions.
  - R-type (*add*, *sub*, *and*, *or*, *slt*)
  - load word (*lw*)
  - store word (*sw*)
  - branch equal (*beq*)
  - jump (*j*)
- The initial steps of a multicycle instruction execution will be similar, but the remaining steps vary depending upon the instruction type.

Steps for an R-Type Instruction

- An instruction is fetched from instruction memory and the PC is incremented.
- Reads two source register values from the register file.
- Performs the ALU operation on the register data operands.
- Writes the result of the ALU operation to the register file.

Steps for a Load Instruction

- An instruction is fetched from instruction memory and the PC is incremented.
- Reads a source register value from the register file and sign extend the 16 least significant bits of the instruction.
- Performs the ALU operation that computes the sum of the value in the register and the sign-extended immediate value from the instruction.
- Accesses data memory at the address of the sum from the ALU.
- Writes the result of the memory value to the register file.

Steps for a Store Instruction

- An instruction is fetched from instruction memory and the PC is incremented.
- Reads two source register values from the register file and sign extend the 16 least significant bits of the instruction.
- Performs the ALU operation that computes the sum of the value in the register and the sign-extended immediate value from the instruction.
- Updates data memory at the address of the sum from the ALU.

Steps for a Branch Equal Instruction

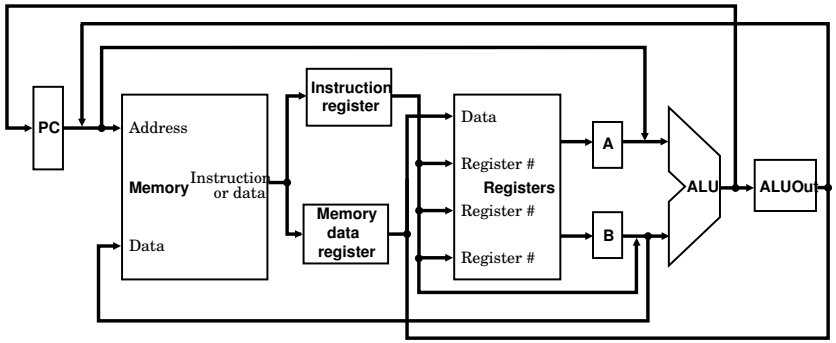
- An instruction is fetched from instruction memory and the PC is incremented.
- Reads two source register values from the register file and sign extends the 16 least significant bits of the instruction and then left shifts it by two.
- The ALU performs a subtract on the data values read from the register file. The value of PC+4 is added with the sign-extended left-shifted by two immediate value from the instruction, which results in the branch target address.
- The Zero result from the ALU is used to decide which adder result to store into the PC.

Steps for a Jump Instruction

- An instruction is fetched from instruction memory and the PC is incremented.
- Concatenates the four most significant bits of the PC, the 26 least significant bits of the instruction, and two zero bits. Assigns the result to the PC.

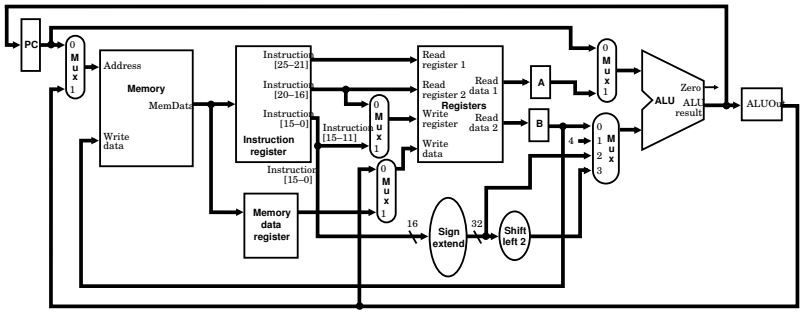
High-Level View of the Multicycle Datapath

- The figure below shows the key elements of the multicycle datapath.
- Performing an instruction across multiple cycles requires new temporary registers to hold data between clock cycles.



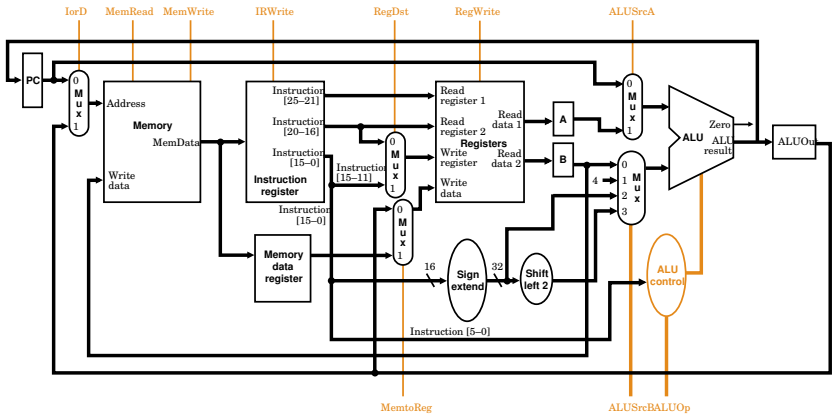
Multicycle Datapath with Some More Detail

- The use of shared functional units requires the addition or widening of multiplexors.
- Now some multiplexors, sign extension unit, shift left 2 unit, and other details have been added.
- Note the multicycle implementation allows two adders and a separate memory unit to be removed.



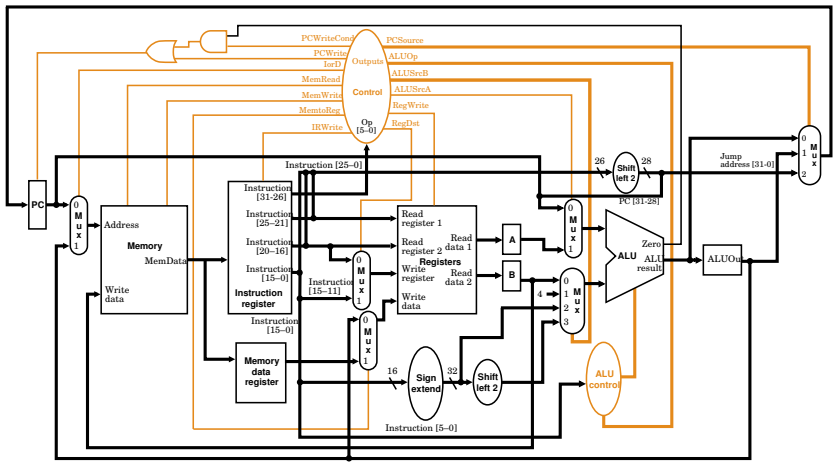
Multicycle Datapath with Control Lines Shown

- The ALUSrcB and ALUOp signals are 2 bits and all other control signals are 1 bit.
- The instruction register has a write signal as it should not be written every cycle and the value from memory can change.



Complete Datapath for the Multicycle Implementation

- Extra control lines and logic gates to control whether or not to update the PC and an extra multiplexor to control which value the PC gets assigned.



Actions of the 1-Bit Control Signals

Signal Name	Effect When Deasserted	Effect When Asserted
RegDst	The register file destination number for the *Write register* comes from the *rt* field.	The register file destination number for the *Write register* comes from the *rd* field.
RegWrite	None	The general-purpose register selected by the *Write register* number is written with the value of the *Write data* input.
ALUSrcA	The first ALU operand is the *PC*.	The first ALU operand comes from the *A* register.
MemRead	None	Content of memory at the location specified by the *Address* input is put on the Memory data output.
MemWrite	None	Memory contents of the location specified by the *Address* input is replaced by the value on the *Write data* input.
MemtoReg	The value fed to the register file input comes from *ALUOout*.	The value fed to the register file input comes from *Memory data register*.
IorD	The *PC* is used to supply the address to the memory unit.	*ALUOut* is used to supply the address to the memory unit.
IRWrite	None	The output of the memory is written into the *Instruction Register* (IR).
PCWrite	None	The *PC* is written; the source is controlled by *PCSource*.
PCWriteCond	None	The *PC* is written if the *Zero* output from the ALU is also active.

Actions of the 2-Bit Control Signals

Signal Name	Value	Effect
ALUOp	00	The ALU performs an add operation.
	01	The ALU performs an subtract operation.
	10	The *funct* field of the instruction determines the ALU operation.
ALUSrcB	00	The second input to the ALU comes from the *B* register.
	01	The second input to the ALU is the constant 4.
	10	The second input to the ALU is the sign-extended, lower 16 bits of the *Instruction Register* (IR).
	11	The second input to the ALU is the sign-extended, lower 16 bits of the IR shifted left by 2 bits.
PCSource	00	Output of the ALU (*PC* + 4) is sent to the *PC* for writing.
	01	The contents of *ALUOut* (the branch target address) are sent to the *PC* for writing.
	10	The jump target address (IR[25-0] shifted left 2 bits and concatenated with PC + 4[31-28]) is sent to the *PC* for writing.

Multicycle Implementation Steps of Execution

- Instruction Fetch Step
- Instruction Decode and Register Fetch Step
- Execution, Memory Address Computation, Branch Completion Step, or Jump Completion Step
- Memory Access or R-Type Instruction Completion Step
- Memory Read Completion Step

Summary of Instruction Steps

- Instructions take from 3 to 5 execution steps.
- First two steps are independent of instruction type.
- A new instruction starts just after the current instruction completes.

Step Name	Actions for Instruction Type			
	R-Type	Memory Reference	Branches	Jumps
Inst Fetch	IR = Mem[PC] PC = PC + 4			
Inst Decode, Reg Fetch, and Branch Calc	A = Reg[IR[25-21]] B = Reg[IR[20-16]] ALUOut = PC + (sign-extend (IR[15-0] << 2)			
R-Type Exec, Address Calc, Branch Compl, or Jump Compl	ALUOut = A op B	ALUOut = A + sign-extend (IR[15-0])	if (A == B) then PC = ALUOut	PC = PC[31-28] || (IR[25-0] << 2)
R-Type Compl or Memory Ref	Reg[IR[15-11] = ALUOut	Load: MDR = Mem[ALUOut] or Store: Mem[ALUOut] = B		
Memory Read Compl		Load: Reg[IR[20-16]] = MDR		

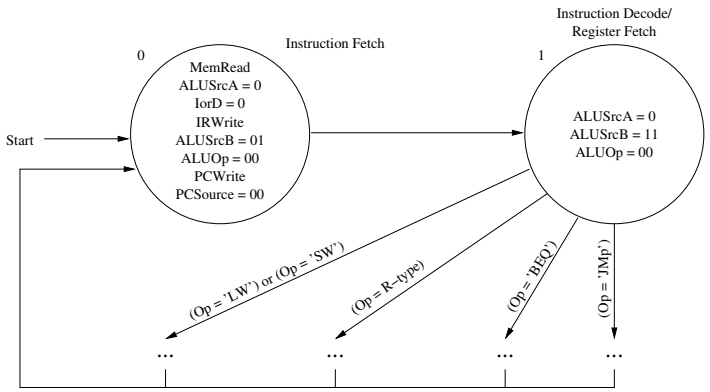
Multicycle Control

- Control in a multicycle implementation must:
  - Specify the signals to be set in each step.
  - Specify the next step to be performed.
- General techniques for specifying control include:
  - **finite state machines**
  - microprogramming

Finite State Machine for Multicycle Control

- set of states (steps of execution)
  - set of inputs (opcode in current instruction, etc.)
  - next-state function (based on opcode)
  - set of outputs (signals to control the processor)

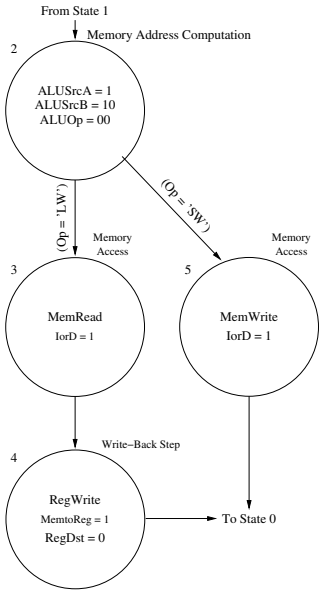
Initial Portion of FSM Control for MIPS



- State 0
  - MemRead, IorD=0, IRWrite: IR=Mem[PC]
  - ALUSrcA=0, ALUSrcB=01, ALUOp=00, PCWrite, PCSource=00: PC=PC+4
- State 1
  - ALUSrcA=0, ALUSrcB=11, ALUOp=00: ALUOut=PC+(sign-extend(IR[15-0])≪2)

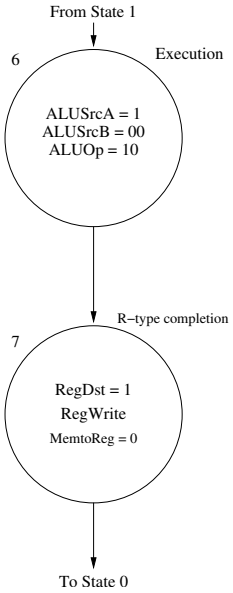
FSM Control for Memory Reference Instructions

- State 2
  - ALUSrcA=1, ALUSrcB=10, ALUOp=00: ALUOut= A+sign-extend(IR[15-0])
- State 3
  - MemRead, IorD=1: MDR=Mem[ALUOut]
- State 4
  - RegWrite, MemtoReg=1, RegDst=0: Reg[IR[20-16]]=MDR
- State 5
  - MemWrite, IorD=1: Mem[ALUOut]=B

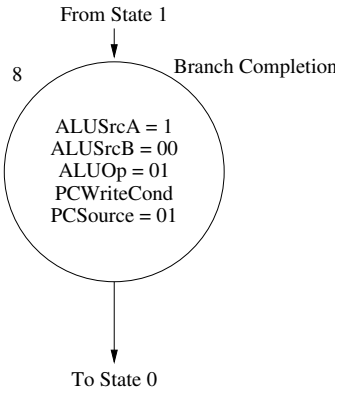


FSM Control for R-Type Instructions

- State 6
  - ALUSrcA=1, ALUSrcB=00, ALUOp=10: ALUOut=A op B
- State 7
  - RegDst=1, RegWrite, MemtoReg=0: Reg[IR[15-11]]=ALUOut

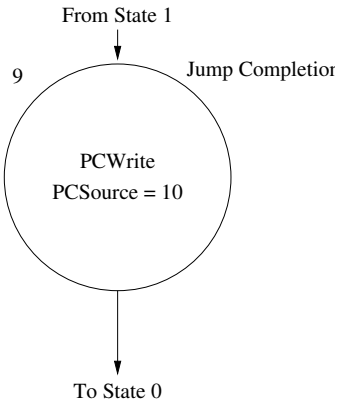


FSM Control for Branch Instructions



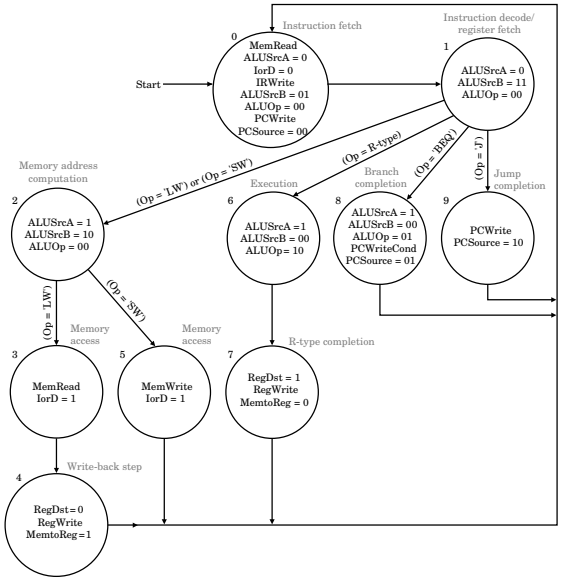
- State 8
  - ALUSrcA=1, ALUSrcB=00, ALUOp=01, PCWriteCond, PCSource=01: if (A == B) then PC=ALUOut

FSM Control for Jump Instructions

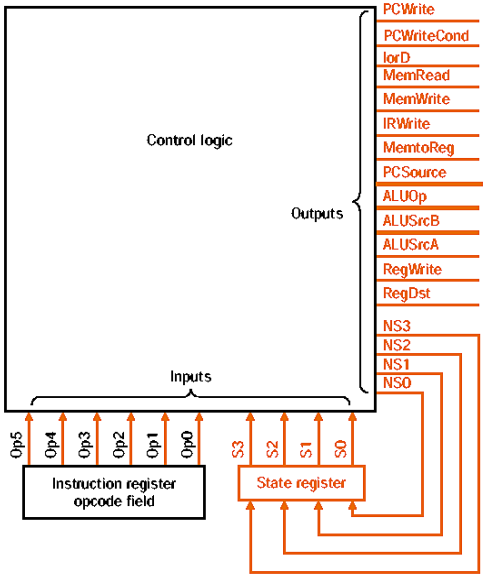


- State 9
  - PCWrite, PCSource=10: PC = PC[31-28] || (IR[25-0]«2)

Complete Finite State Machine for MIPS Control



Finite State Machine Control Unit for MIPS



Advantages/Disadvantages of a Multicycle Implementation

- advantages
  - Can significantly reduce execution time as compared to the single cycle implementation.
    - The clock cycle can be much shorter as it is dependent on the longest stage in the multicycle datapath.
    - Allows different instruction types to require a varying number of cycles depending on the steps that need to be performed.
  - Can require less hardware since all units are not shared during a single clock cycle.
    - Can use a single memory for instructions and data.
    - Can eliminate some adders.
- disadvantages
  - Implementation of control a little more complex.
  - Longest instruction takes a little longer.