COMPUTER ORGANIZATION (IS F242)

LECT 33_34: MIPS AND CACHE MEMORY

- We modify the data path for including 2 exceptions
  - Undefined instruction
  - Arithmetic overflow
- Action to perform when exception occurs/detected
  - The processor's control circuitry must be able to
    - Save the address of the instruction that caused the exception in the exception counter (EPC)
    - Transfer control to OS at a pre-specified address
- OS takes appropriate action (*exception handling)*.
  - Provide some service to user program, taking some predefined action in response to an overflow or stopping the execution of the program and reporting an error
  - OS may continue or terminate the program after exception (EPC is used to determine where to restart the execution)

- How OS will know the reason for exception?
  - 2 methods possible
  - Include a status register (MIPS uses this method)
    - Cause register is used in MIPS which holds a field that indicates the reason for the exception
    - Single entry point for all exceptions. OS decodes the status register to find the cause
  - Use vectored interrupts
    - Address to which control transfers is determined by the cause of the exception
    - Example: Undefined instruction (C000 0000) and Arithmetic overflow (C000 0020)
    - Address is separated by 32 bytes (8 instructions) – OS will perform some limited processing in this sequence

Hardware support for Exception

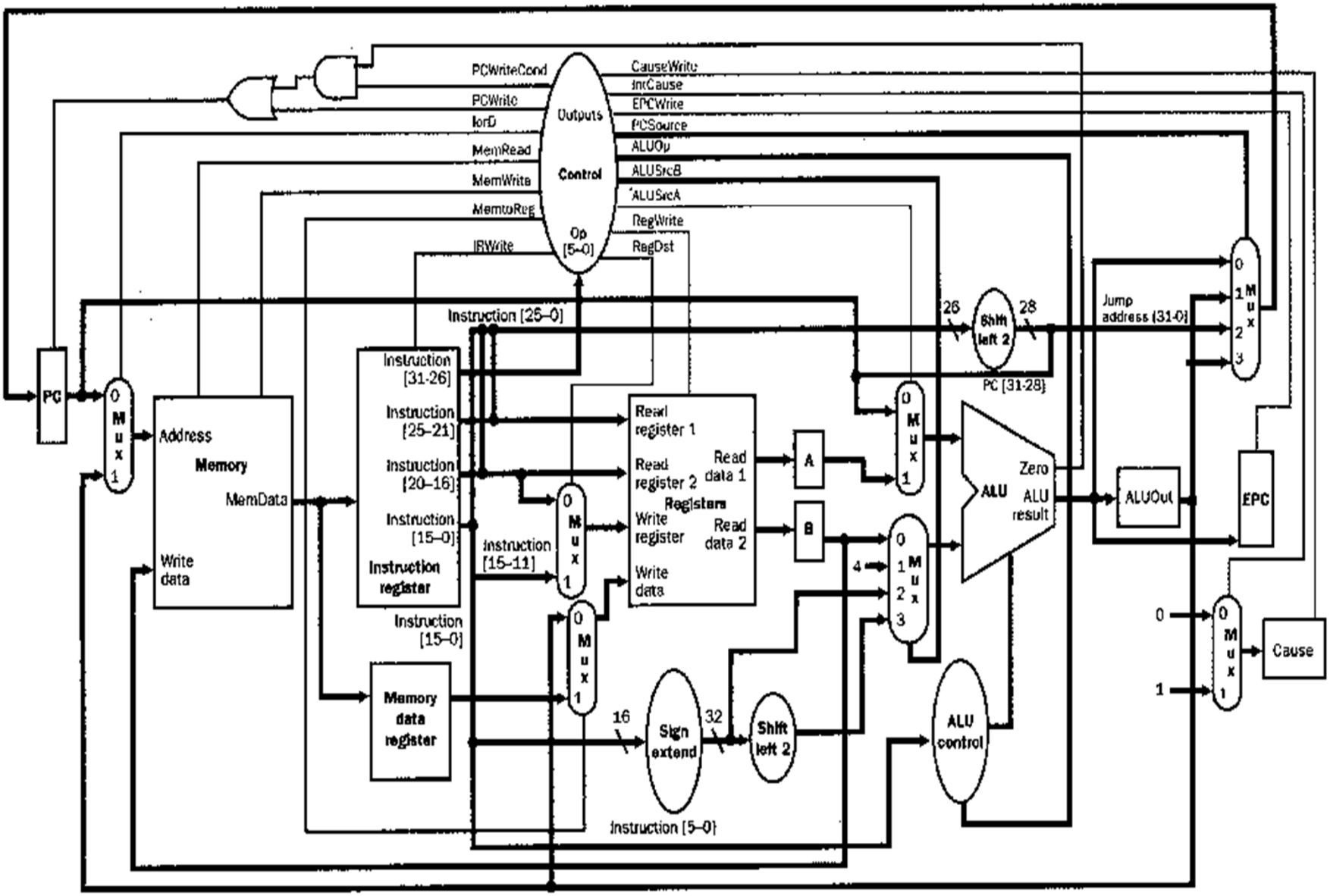
- *EPC*:
  - 32-bit register holds the address of the affected instruction
- *Cause*:
  - 32-bit register contains a binary code that describes the cause or type of exception
  - Assume the low order bit of this register encodes the two possible exception sources
    - Undefined instruction = 0
    - Arithmetic overflow = 1

Additional control signals for Exception

- EPCWrite
  - EPC register to be written
- CauseWrite
  - Cause register to be written
- IntCause
  - 1 – bit control signal to set the low order bit of the Cause register appropriately
- Need to write Exception address (OS entry point for exception handling) into PC
  - In MIPS this address is 8000 0180

Modification in data path

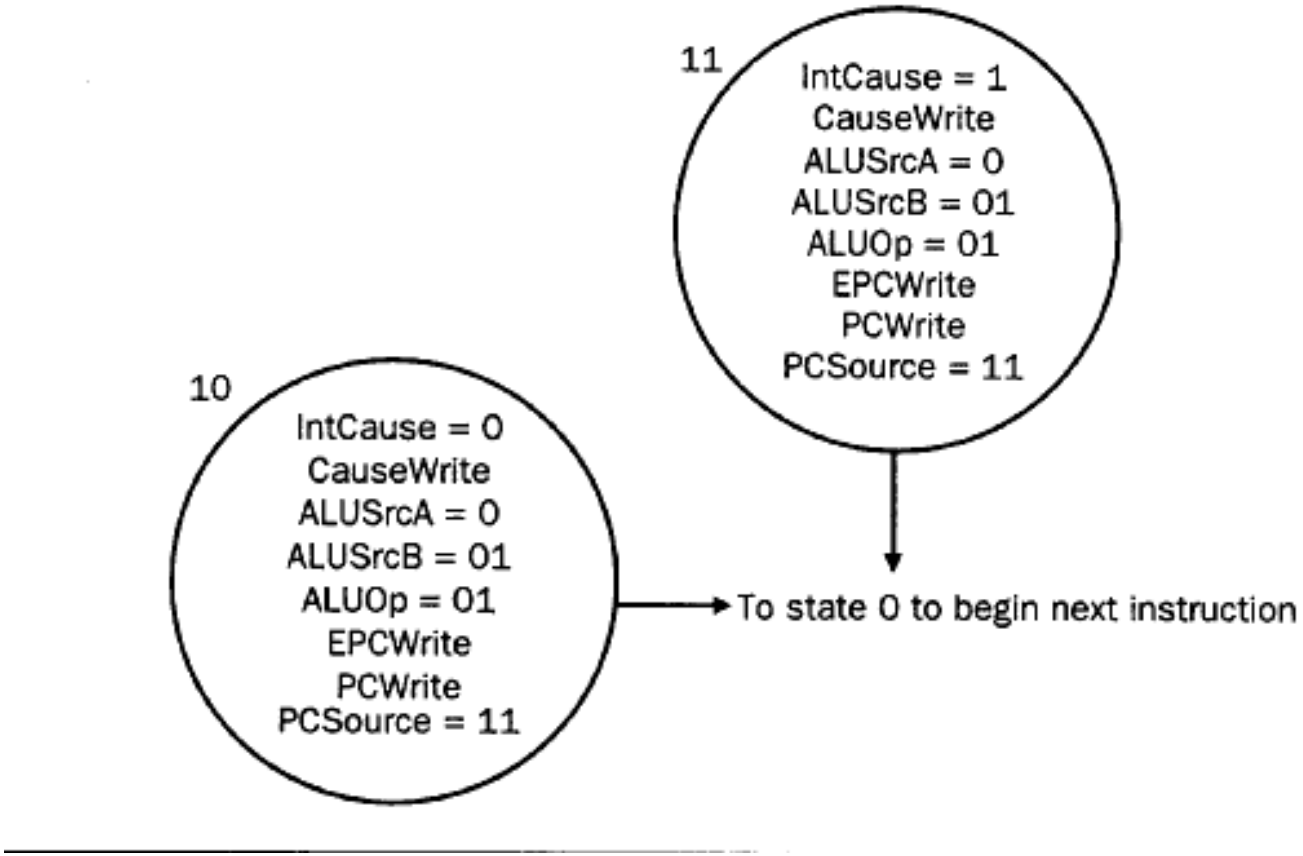
- Make MUX with PCSource control signal as 4:1 MUX
  - PCSource = 11 will write 8000 0180 to PC
- EPC should store current PC -4 as PC would have been incremented in first cycle
  - Use ALU for subtraction as PC and 4 are inputs for ALU


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5–0]
CauseWrite
IntCause
EPCWrite
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
Instruction [25–0]
26
Shift left 2
28
PC [31-28]
Jump address [31-0]
PC
Address
Memory
MemData
Write data
Instruction [31-26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–0]
Instruction register
Instruction [15–0]
Instruction [15–11]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
A
B
4
ALU
Zero
ALU result
ALUOut
EPC
Cause
Memory data register
16
Sign extend
32
Shift left 2
ALU control
Instruction [5–0]

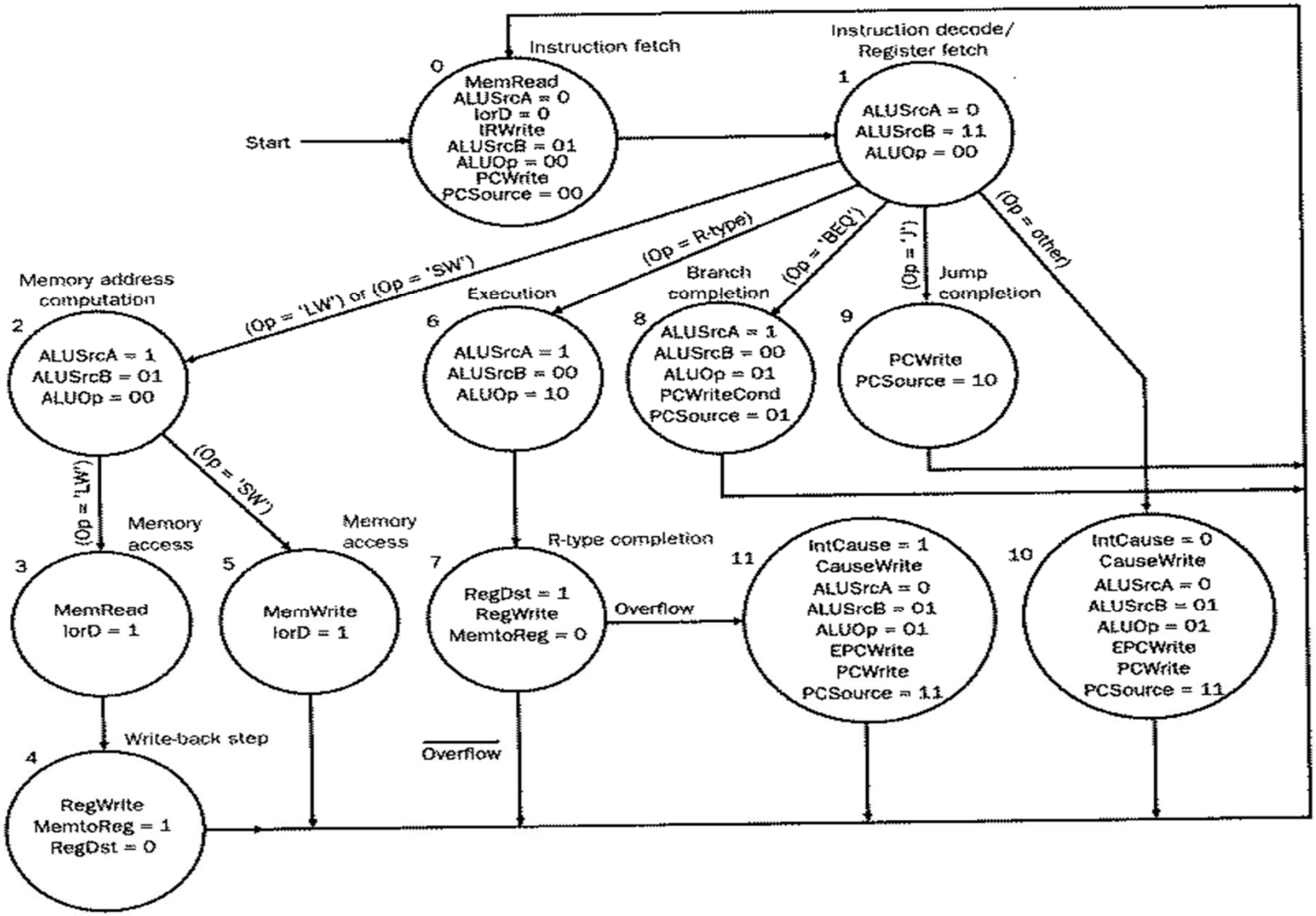
Exception detection

- *Undefined Instruction*:
  - Finite state control must be modified to define the next-state value as 10 for all operation types other than the five that are allowed (i.e., lw, sw, beq, jump, and R-format)
- *Arithmetic Overflow*:
  - ALU can be designed to include overflow detection logic with a signal output from the ALU called *overflow*, which is asserted if overflow is detected. This is used to specify the next state for State 7

New states for Exception



FSM with Exception



Problems with Exception handling in MIPS

- Rollback and Restart
  - *Rollback*: Reverse effects of the process
  - *Restart*: Do a process or operation over again
  - Overflow exception detected *after* ALUout written
  - This means ALU operation produced an uncorrected error
  - Current FSC does not support process restart or rollback
- How to Fix?
  - Not easy – Control system design is difficult
    - – Requires much additional HW to do rollback

Microprogramming

- FSM is the best option if the number of states and number of sequences are less
- When the number of states are large
  - use some of the ideas from programming to help create a method of specifying the control that will make it easier to understand as well as to design

Microprogramming

- Say a set of control signals that must be asserted in a state as an instruction to be executed by the datapath
  - To avoid confusion with MIPS instruction set we call these instructions as microinstructions
  - Each microinstruction defines the set of datapath control signals that must be asserted in a given state
  - Executing a microinstruction has the effect of asserting the control signals specified by the microinstruction.

Microprogramming

- Designing the control as a program that implements the machine instructions in terms of simpler microinstructions is called microprogramming
- Represent a microinstruction syntactically as a sequence of fields whose functions are related
- Microprogram uses the concept of sub-routines to improve reuse
- A *microprogram is a sequence of microinstructions (*Abstraction of datapath control)

each microinstruction has eight fields (label + 7 functional)

Field	Specifies
•*Label*	Used to control microcode sequencing
•*ALU control*	ALUop for this clock cycle
• *SRC1*	Source for first ALU operand
• *SRC2*	Source for second ALU operand
• *Register Control*	Register read/write, data value written
• *Memory*	Memory read/write, data value written Destination register for MemRead
• *PCWrite control*	Source for PC (PC+4, BTA, JTA)
• *Sequencing*	How to choose next microinstruction

Field name	Values for field	Function of field with specific value
Label	Any string	Used to specify labels to control microcode sequencing. Labels that end in a 1 or 2 are used for dispatching with a jump table that is indexed based on the opcode. Other labels are used as direct targets in the microinstruction sequencing. Labels do not generate control signals directly but are used to define the contents of dispatch tables and generate control for the Sequencing field.
ALU control	Add	Cause the ALU to add.
	Subt	Cause the ALU to subtract; this implements the compare for branches.
	Func code	Use the instruction's funct field to determine ALU control.
SRC1	PC	Use the PC as the first ALU input.
	A	Register A is the first ALU input.
SRC2	B	Register B is the second ALU input.
	4	Use 4 for the second ALU input.
	Extend	Use output of the sign extension unit as the second ALU input.
	Extshft	Use the output of the shift-by-two unit as the second ALU input.
Register control	Read	Read two registers using the rs and rt fields of the IR as the register numbers, putting the data into registers A and B.
	Write ALU	Write the register file using the rd field of the IR as the register number and the contents of ALUOut as the data.
	Write MDR	Write the register file using the rt field of the IR as the register number and the contents of the MDR as the data.
Memory	Read PC	Read memory using the PC as address; write result into IR (and the MDR).
	Read ALU	Read memory using ALUOut as address; write result into MDR.
	Write ALU	Write memory using the ALUOut as address; contents of B as the data.
PCWrite control	ALU	Write the output of the ALU into the PC.
	ALUOut-cond	If the Zero output of the ALU is active, write the PC with the contents of the register ALUOut.
	Jump address	Write the PC with the jump address from the instruction.
Sequencing	Seq	Choose the next microinstruction sequentially.
	Fetch	Go to the first microinstruction to begin a new instruction.
	Dispatch i	Dispatch using the ROM specified by i (1 or 2).

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1

Fields	Effect
ALU control, SRC1, SRC2	Compute PC + 4. (The value is also written into ALUOut, though it will never be read from there.)
Memory	Fetch instruction into IR.
PCWrite control	Causes the output of the ALU to be written into the PC.
Sequencing	Go to the next microinstruction.

Fields	Effect
ALU control, SRC1, SRC2	Store PC + sign extension (IR[15:0]) << 2 into ALUOut.
Register control	Use the rs and rt fields to read the registers placing the data in A and B.
Sequencing	Use dispatch table 1 to choose the next microinstruction address.

Microprogramming

- Dispatch operation can be a switch *case* statement with the opcode field and the dispatch table 1 used to select one of four different microinstruction sequences with one of four different labels (all ending in “1”):
- Mem1
  - for memory-reference instructions
- Rformat1
  - for R-type instructions
- BEQ1
  - for the branch equal instruction
- JUMP1
  - for the jump instruction

Mem1

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch

Fields	Effect
ALU control, SRC1, SRC2	Compute the memory address: Register (rs) + sign-extend (IR[15:0]), writing the result into ALUOut.
Sequencing	Use the second dispatch table to jump to the microinstruction labeled either LW2 or SW2.

Fields	Effect
Memory	Read memory using the ALUOut as the address and writing the data into the MDR.
Sequencing	Go to the next microinstruction.

Fields	Effect
Register control	Write the contents of the MDR into the register file entry specified by rt.
Sequencing	Go to the microinstruction labeled Fetch.

Fields	Effect
Memory	Write memory using contents of ALUOut as the address and the contents of B as the value.
Sequencing	Go to the microinstruction labeled Fetch.

Rformat1

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch

Fields	Effect
ALU control, SRC1, SRC2	The ALU operates on the contents of the A and B registers, using the function field to specify the ALU operation.
Sequencing	Go to the next microinstruction.

Fields	Effect
Register control	The value in ALUOut is written into the register file entry specified by the rd field.
Sequencing	Go to the microinstruction labeled Fetch.

BEQ1

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
BEQ1	Subt	A	B			ALUOut-cond	Fetch

Fields	Effect
ALU control, SRC1, SRC2	The ALU subtracts the operands in A and B to generate the Zero output.
PCWrite control	Causes the PC to be written using the value already in ALUOut, if the Zero output of the ALU is true.
Sequencing	Go to the microinstruction labeled Fetch.

JMP1

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
JUMP1						Jump address	Fetch

Fields	Effect
PCWrite control	Causes the PC to be written using the jump target address.
Sequencing	Go to the microinstruction labeled Fetch.

Microprogram for the control unit

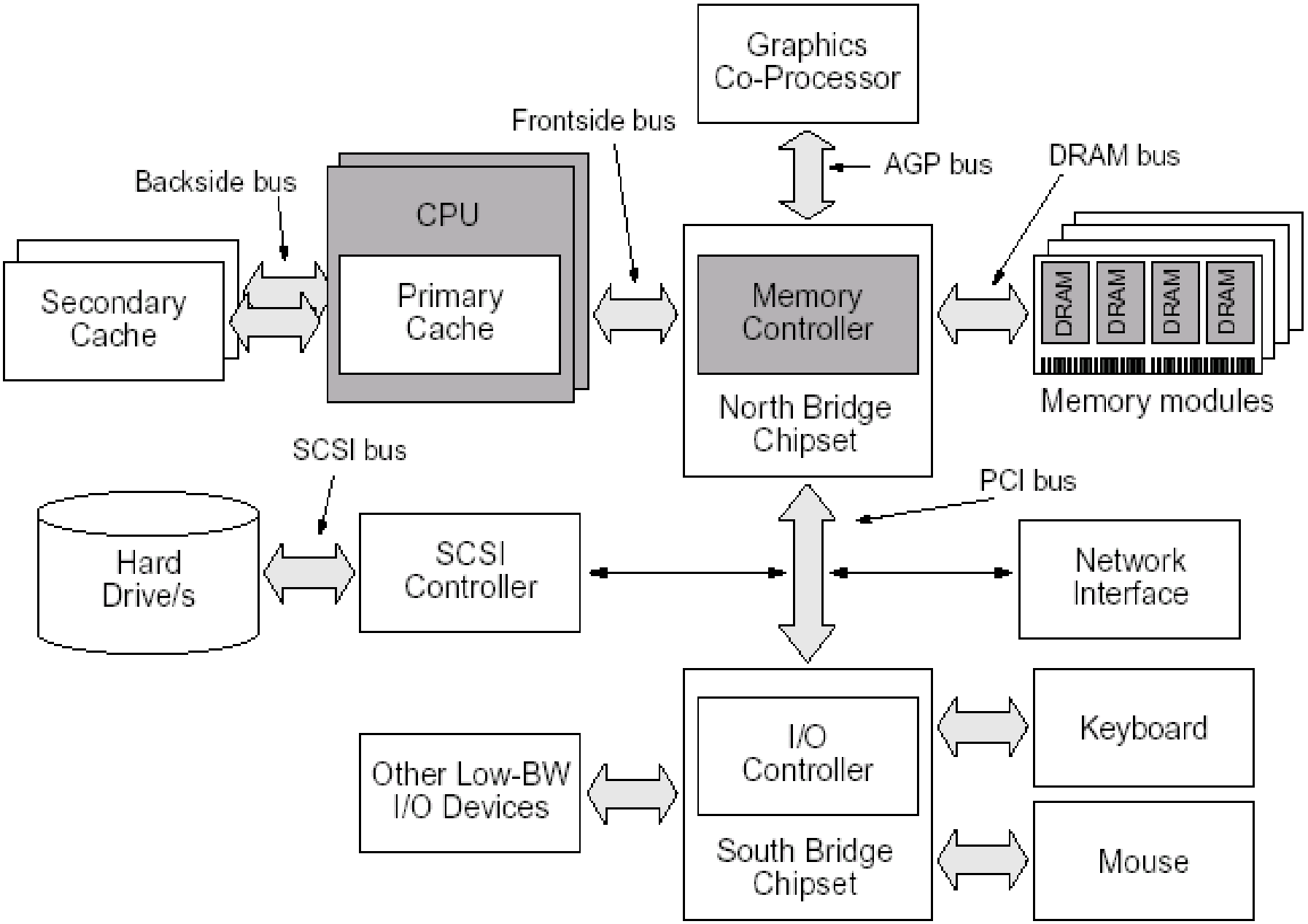
Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch
BEQ1	Subt	A	B			ALUOut-cond	Fetch
JUMP1						Jump address	Fetch

Microcode: Trade-offs

- Specification advantages
  - easy to design and write
  - typically manufacturer designs architecture and microcode in parallel
- Implementation advantages
  - easy to change since values are in memory (e.g., off-chip ROM)
  - can emulate other architectures
  - can make use of internal registers
- Implementation disadvantages
  - control is implemented nowadays on same chip as processor so the advantage of an off-chip ROM does not exist
  - ROM is no longer faster than on-board cache
  - there is little need to change the microcode as general-purpose computers are used far more nowadays than computers designed for specific applications

Memory

- Characteristics
  - Properties
  - Location
  - Capacity
  - Unit of transfer
  - Access method
  - Performance
  - Organisation
  - Semiconductor Memories


Graphics
Co-Processor
Frontside bus
AGP bus
DRAM bus
Backside bus
CPU
Secondary
Cache
Primary
Cache
Memory
Controller
DRAM
DRAM
DRAM
DRAM
North Bridge
Chipset
Memory modules
SCSI bus
PCI bus
Hard
Drive/s
SCSI
Controller
Network
Interface
I/O
Controller
Keyboard
Other Low-BW
I/O Devices
South Bridge
Chipset
Mouse

Properties

- Inclusive
- Coherence
- Locality of reference
  - Program access a relatively small portion of the address space at any instant of time
  - Two different types
    - Spatial (in space)
      - if an item is referenced, items close by tend to be referenced soon
    - Temporal (in time)
      - if an item is referenced, it will tend to be referenced again soon

Location

- CPU
- Internal
- External

Capacity

- Word size
  - The natural unit of organisation
- Number of words
  - or Bytes

Unit of Transfer

- Internal
  - Usually governed by data bus width
- External
  - Usually a block which is much larger than a word
- Addressable unit
  - Smallest location which can be uniquely addressed
  - Word internally
  - Cluster on Hard disks

Access Methods (1)

- Sequential
  - Start at the beginning and read through in order
  - Access time depends on location of data and previous location
  - e.g. tape
- Random
  - Individual addresses identify locations exactly
  - Access time is independent of location or previous access
  - e.g. RAM

Access Methods (2)

- Direct
  - Individual blocks have unique address
  - Access is by jumping to vicinity plus sequential search (Random+Sequential)
  - Access time depends on location and previous location e.g. disk
- Associative
  - Data is located by a comparison with contents of a portion of the store
  - Access time is independent of location or previous access
  - e.g. cache

Performance

- Access time (t_a)
  - Time between presenting the address and getting the valid data
- Memory Cycle time (t_c)
  - Time may be required for the memory to “recover” before next access
    - Memory needs address lines to be stable between accesses
  - Cycle time = access time + recovery time
- Transfer Rate (b) $b = w / t_c$ bps
  - Rate at which data can be moved