Datapath III -- Microprogramming

Example

- Which of the following implementations would be faster and by how much
  - Implementation which uses a fixed length clock cycle
  - Implementation where clock cycle length determined by instruction
- Assume:
  - Memory units have 2ns delay
  - ALU and adders have 2ns delay
  - Register file has 1ns delay
  - All other units have 0 delay
- Assume following instruction mix:
  - 24% loads
  - 12% stores
  - 44% R-Type
  - 18% branches
  - 2% Jumps

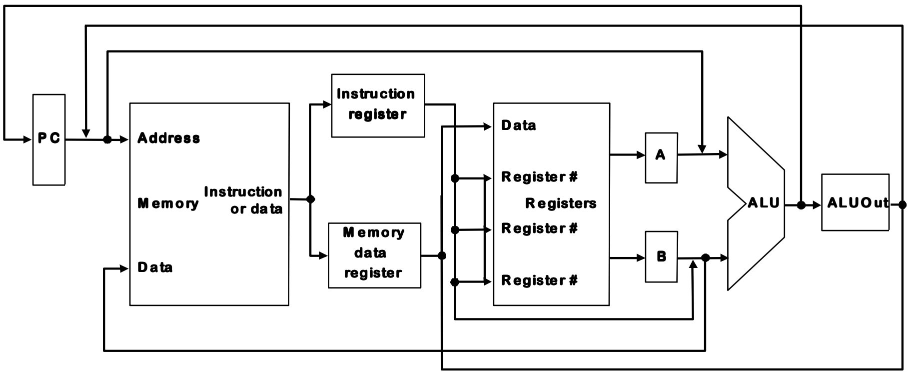
Example

- Timing results
  - Loads: 8ns
  - Stores: 7ns
  - R-type: 6ns
  - Branch: 5ns
  - Jump: 2ns
- (8 x 24%) + (7 x 12%) + (6 x 44%) + (5 x 18%) + (2 x 2%) = 6.3ns
- 8/6.3 = 1.27 times faster

MultiCycle Implementation

- Instruction execution can be broken into the following steps
  - Instruction Fetch
  - Instruction Decode
  - Operand Fetch
  - Execute
  - Store Results
- Also allows sharing of components
  - single memory for data and instructions
  - Single ALU
- Extra registers required to store results between stages

High Level Datapath



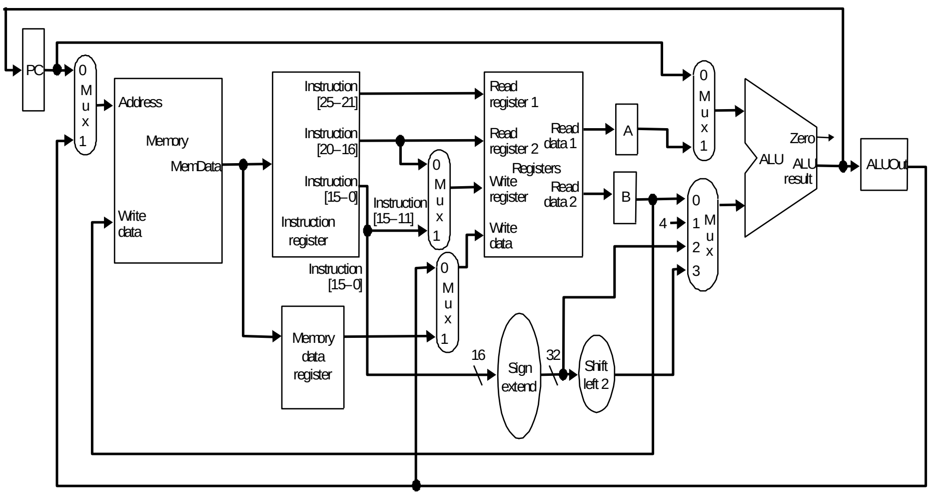
Additional Registers

- Determined by:
  - What combination units can fit in a clock cycle
  - What data is required in later clock cycles
- Assume at most one of the following can be accommodated by a single clock cycle:
  - A memory access
  - A register file access (two reads or one write)
  - An ALU operation
- Require the following additional registers
  - IR
  - MDR
  - A and B registers
  - ALUout register
- All new regs except IR only have to hold values from one clock cycle to next so do not require write control signal

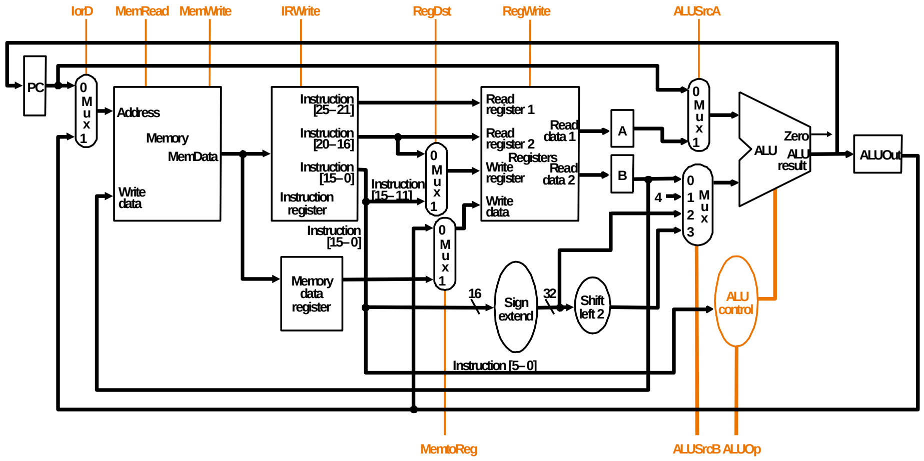
Combining ALUs

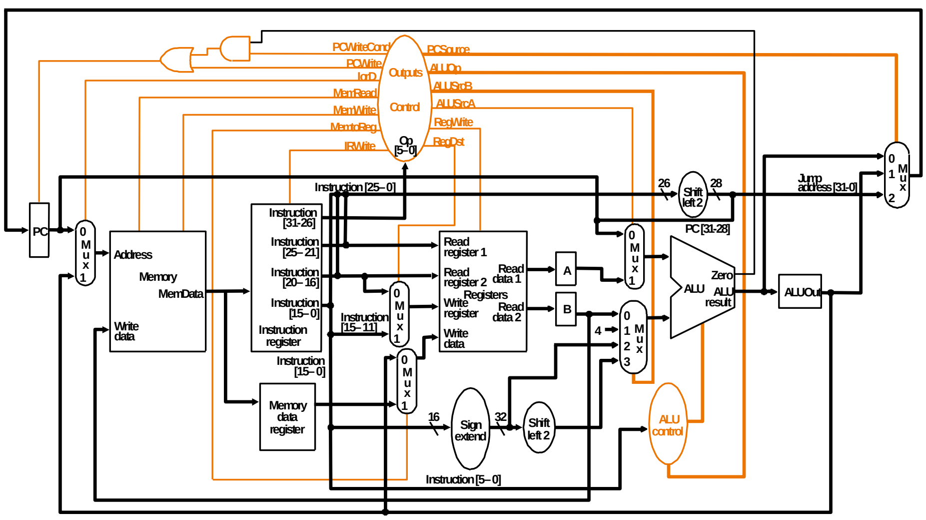
- Merging all ALUs and adders into a single datapath requires two main changes:
  - An additional multiplexer on the A input to the ALU, selects:
    - PC
    - A register
  - Extend multiplexer on second input
    - The constant 4 for incrementing PC
    - The sign extended and shifted 16-bit offset field used in branch address computation

Datapath

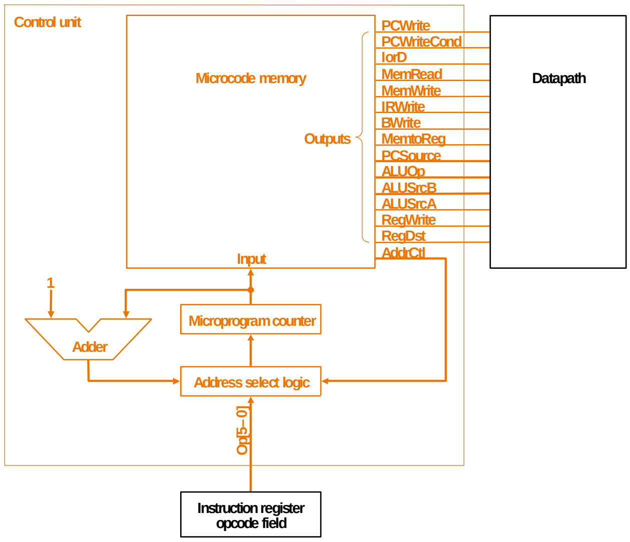


Datapath with Control Signals





Microprogramming



What are the "microinstructions" ?

Microprogramming

- A specification methodology
  - appropriate if hundreds of opcodes, modes, cycles, etc.
  - signals specified symbolically using microinstructions

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

- *Will two implementations of the same architecture have the same microcode?*
- *What would a microassembler do?*

Microinstruction format

Field name	Value	Signals active	Comment
ALU control	Add	ALUOp = 00	Cause the ALU to add.
	Subt	ALUOp = 01	Cause the ALU to subtract; this implements the compare for branches.
	Func code	ALUOp = 10	Use the instruction's function code to determine ALU control.
SRC1	PC	ALUSrcA = 0	Use the PC as the first ALU input.
	A	ALUSrcA = 1	Register A is the first ALU input.
SRC2	B	ALUSrcB = 00	Register B is the second ALU input.
	4	ALUSrcB = 01	Use 4 as the second ALU input.
	Extend	ALUSrcB = 10	Use output of the sign extension unit as the second ALU input.
	Extshft	ALUSrcB = 11	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 and putting the data into registers A and B.
	Write ALU	RegWrite, RegDst = 1, MemtoReg = 0	Write a register using the rd field of the IR as the register number and the contents of the ALUOut as the data.
	Write MDR	RegWrite, RegDst = 0, MemtoReg = 1	Write a register using the rt field of the IR as the register number and the contents of the MDR as the data.
Memory	Read PC	MemRead, lorD = 0	Read memory using the PC as address; write result into IR (and the MDR).
	Read ALU	MemRead, lorD = 1	Read memory using the ALUOut as address; write result into MDR.
	Write ALU	MemWrite, lorD = 1	Write memory using the ALUOut as address, contents of B as the data.
PC write control	ALU	PCSource = 00 PCWrite	Write the output of the ALU into the PC.
	ALUOut-cond	PCSource = 01, PCWriteCond	If the Zero output of the ALU is active, write the PC with the contents of the register ALUOut.
	jump address	PCSource = 10, PCWrite	Write the PC with the jump address from the instruction.
Sequencing	Seq	AddrCtl = 11	Choose the next microinstruction sequentially.
	Fetch	AddrCtl = 00	Go to the first microinstruction to begin a new instruction.
	Dispatch 1	AddrCtl = 01	Dispatch using the ROM 1.
	Dispatch 2	AddrCtl = 10	Dispatch using the ROM 2.

Maximally vs. Minimally Encoded

- No encoding:
  - 1 bit for each datapath operation
  - faster, requires more memory (logic)
  - used for Vax 780 — an astonishing 400K of memory!
- Lots of encoding:
  - send the microinstructions through logic to get control signals
  - uses less memory, slower
- Historical context of CISC:
  - Too much logic to put on a single chip with everything else
  - Use a ROM (or even RAM) to hold the microcode
  - It’s easy to add new instructions

Microcode: Trade-offs

- Distinction between specification and implementation is sometimes blurred
- Specification Advantages:
  - Easy to design and write
  - Design architecture and microcode in parallel
- Implementation (off-chip ROM) Advantages
  - Easy to change since values are in memory
  - Can emulate other architectures
  - Can make use of internal registers
- Implementation Disadvantages, SLOWER now that:
  - Control is implemented on same chip as processor
  - ROM is no longer faster than RAM
  - No need to go back and make changes

Summary

- Can reduce the length of time some classes of instructions take to execute by moving to a multi-cycle implementation.
- CPI for multi-cycle implementation will be > 1
- But: goal is to maximise the number of instructions executed per clock cycle
- Next week:
  - Pipelining
  - Hazards