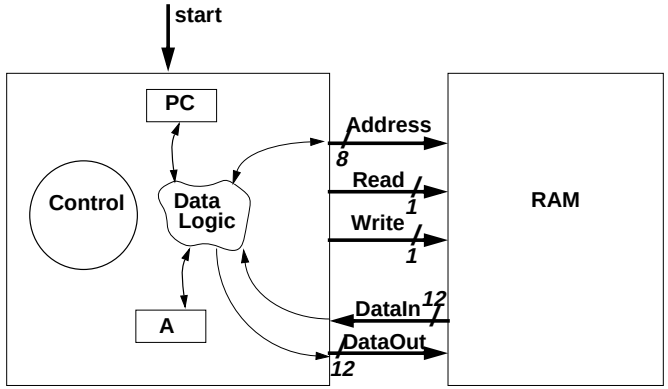
Unit 4

A Simple Processor
Simple-12

What is Simple-12?

- Expansion of MaxFinder into a simple computer
- Reminiscent of first true computers
- Focus on Micro operations, Micro Instructions, Microprograms
- Gain some experience with design-induced limitations

SIMPLE-12 Basic Organization



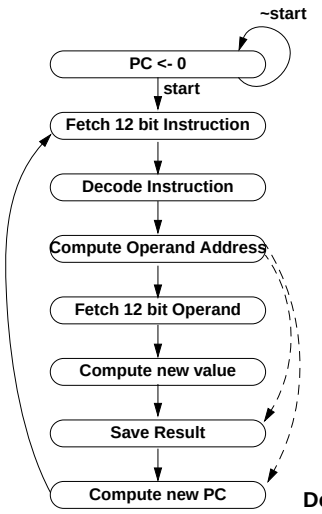
Programmer Visible Registers:

A: Accumulator: - Holds 12 data bits

PC: Program Counter- Holds 8 bits of current instruction address

Question: Is this a Harvard or Princeton machine?

Simple-12: Skeleton Microprogram



Instruction Format:

Opcode (4)	Address (8)

Machine general properties

- 256 words of 12 bits each
- Addresses all 8 bits wide
- Opcode selects one of 16 instructions from 4 Classes:
  - Jump Class: change PC
  - Load/Store Class: access memory
  - Operate Class: compute new value
  - Reserved (for your later enjoyment)

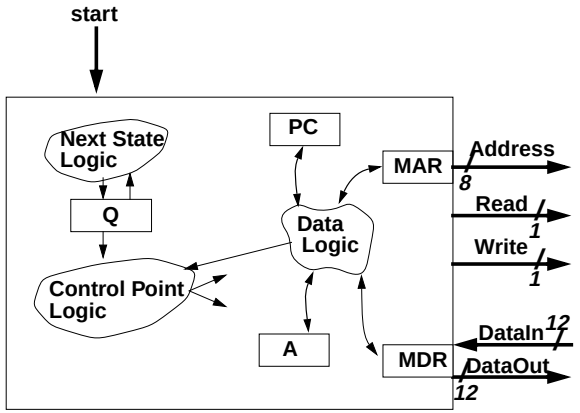
Dotted lines represent instructions which do not need certain micro-instructions

Simple-12 Instruction Set Architecture (ISA)

Opcode	Mnemonic	RTL
0000	JMP X	PC<-X
0001	JN X	if A<0 then PC<-X else PC++
0010	JZ X	if A=0 then PC<-X else PC++
0011	reserved	
0100	LOAD X	A<-M(X) || PC++
0101	STORE X	M(X)<-A || PC++
0110	reserved	
0111	reserved	
1000	AND X	A<- A and M(X) || PC++
1001	OR X	A<- A or M(X) || PC++
1010	ADD X	A<- A + M(X) || PC++
1011	SUB X	A<- A - M(X) || PC++
1100	reserved	
1101	reserved	
1110	reserved	
1111	reserved	

After each instruction completes, next instruction is from Mem(PC)

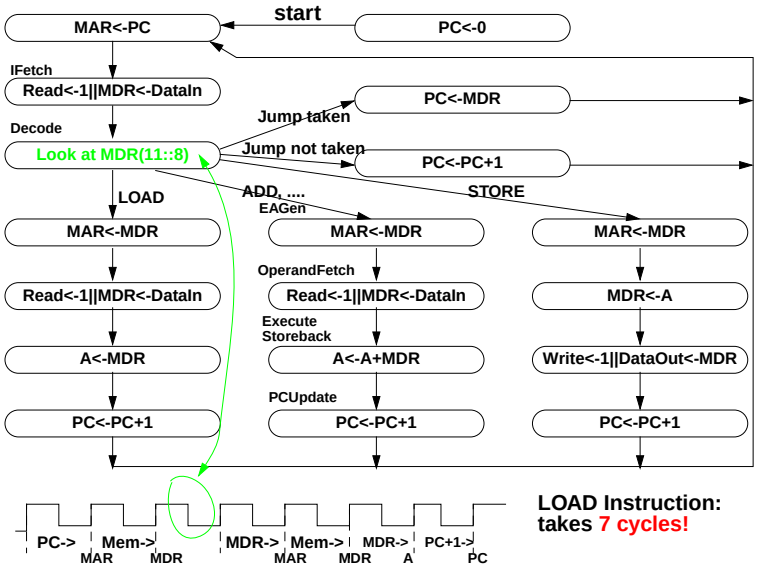
A MaxFinder-Like Organization



Note: Q, MAR, MDR are part of organizational view of SIMPLE-12

Note!!!: Q, MAR, MDR are NOT PART of architecture; They are NOT VISIBILE to the programmer!

A More Detailed Microprogram (Moore Machine)



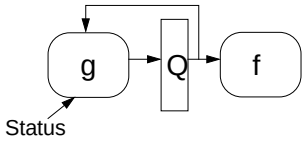
LOAD Instruction: takes 7 cycles!

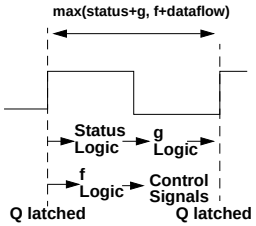
Instruction Cycle Count

Opcode	# Cycles	# Memory Reads	# Memory Writes
JMP X	4	1	0
JN X	4	1	0
JZ X	4	1	0
LOAD X	7	2	0
STORE X	7	1	1
AND X	7	2	0
OR X	7	2	0
ADD X	7	2	0
SUB X	7	2	0

Aside: State Machine Types & Timing

Moore Machine

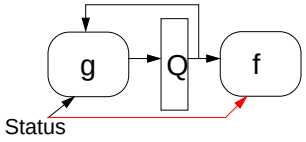


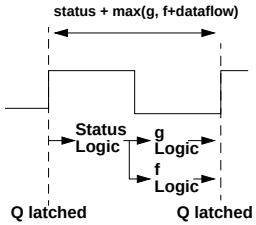


+ Almost all of clock available to dataflow

- Takes 2 cycles for status to affect dataflow

Mealy Machine





+ A status signal can affect dataflow in ssame cycle

- Long clock period

Potential Improvements

- Moore model forbids using instruction being read in from memory at Ifetch from being used in immediate decode
  - Cannot use it in next cycle for control signals because opcode is in MDR, not Q
  - Thus for pure Moore model, we must wait one cycle to get opcode into Q
- Solution: Add a new register IR
  - Loaded from memory bits 11 thru 8 at same time as MDR
  - Serves directly as part of Q state register
- Why waste a cycle in each instruction moving PC to MAR?
  - Include this microoperation at same time that PC is updated
- The only write to memory is from A. Why not tie it to Data_out and save the cycle of transfer MDR<-A?
- We often have a separate cycle PC<-PC+1
  - Why not do this operation in parallel with some other operation
- Most microinstructions are “the same” or “similar” across multiple instruction opcodes
  - Why not group into common states (requires IR as part of Q)
- Several other improvements possible

A Revised Microprogram in RTL

Stopped: If start=1 then (MAR<-PC<-0 || -> IFetch) else -> stopped

IFetch: Read<-1 || MDR<-DataIn || PC<-PC+1 || IR<-DataIn(11:8)||->EAGen

EAGen: if (IR=JMP) or (IR=JN and A(11)=1) or (IR=JZ and A=0)

then (MAR<-PC<-MDR(7:0) || ->IFetch)

else if (IR=JN) or (IR=JZ) then (MAR <-PC || ->IFetch)

else MAR<-MDR(7:0) || ->OperandAccess

Note: Mealy dependency of dataflow on A(11), A=0

OperandAccess: MAR<-PC ||

if IR=LOAD or IR(3:2)=10

then (Read<-1 || MDR<-DataIn || -> Execute)

else (i.e. a STORE) Write<-1 || DataOut<-A || ->IFetch

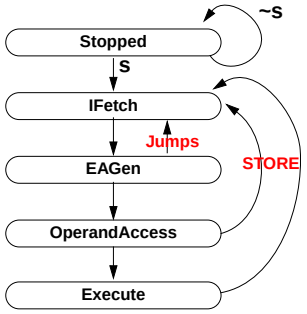
Execute: -> IFetch ||

if IR=LOAD then A<- MDR

else if IR=ADD then ....

Note: If Read or Write are not specified in a state, then they must be 0

Revised Timing

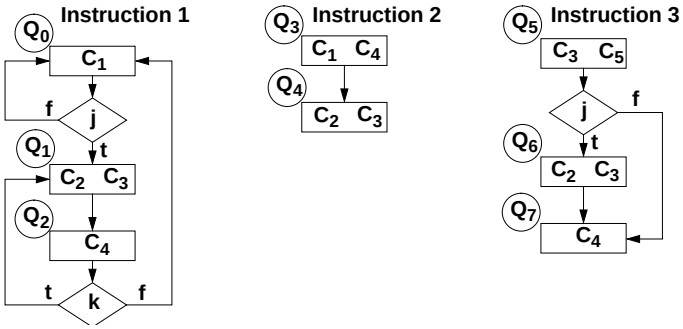


Opcode	Cycles
JMP X	2
JN X	2
JZ X	2
LOAD X	4
STORE X	3
AND X	4
OR X	4
ADD X	4
SUB X	4

Microprogram Controller Design

Objective: Design a controller that interprets each machine language instruction as a microprogram routine.

Insight: Decode machine language op-code to determine starting state (address) of microprogram routine.



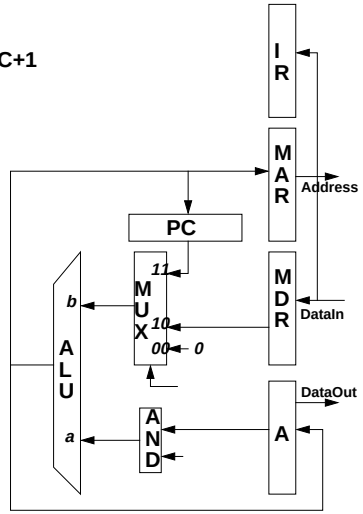
Now Let's Tackle the DataFlow Design

First, what dataflow microoperations have to be done (and concurrently)?

- PC<-0
- MDR<-DataIn || IR<-DataIn || PC<-PC+1
- MAR<-MDR(7:0) || PC <- MDR(7:0)
- MAR<-PC
- A<-MDR
- A<-A+MDR
- A<-A-MDR
- A<-A and MDR
- A<-A or MDR

Resulting Control Points:

- Load Signals for each register:
  - LoadPC, Load MDR, ...
- ALU Controls (4 bits)
  - b-invert, carry-in, op1, op0
- MUX1: Select PC, MDR, 0
- AND: Select 0 (0) or A



Mealy Design Problem

Observation on RTL at State EAGen: we have a Mealy Machine where dataflow operations depend on status signals A(11) and A=0

EAGen: if (IR=JMP) or (IR=JN and A(11)=1) or (IR=JZ and A=0)

then (MAR<-PC<-MDR(7:0) || ->IFetch)

else if (IR=JN) or (IR=JZ) then (MAR <-PC || ->IFetch)

else MAR<-MDR(7:0) || -> OperandAccess

With current dataflow, using ALU zero tester to test A=0, we cannot test A and then change inputs to ALU to either MDR or PC IN SAME CYCLE!

One solution: insert an extra step for jumps not taken.

Control Signal Table

Q	IR	s	A_{11}	=0	Q*	LA	LPC	LMAR	LMDR	LIR	ALU	bMUX	aGate	Read	Write
Stopped		0			Stopped	0	0	0	0	0				0	0
Stopped		1			IFetch	0	1	1	0	0	+	0	0	0	0
Ifetch					EAGen	0	1	0	1	1	+1	PC	0	1	0
EAGen	JMP				IFetch	0	1	1	0	0	+	MDR	0	0	0
EAGen	JN		1		IFetch	0	1	1	0	0	+	MDR	0	0	0
EAGen	JN		0		IFetch	0	0	1	0	0	+	PC	0	0	0
EAGen	JN		0 1		IFetch NoJump	0	0	1	0	0	+	PC		0	0
EAGen	JZ			1	IFetch	0	1	1	0	0	+	MDR	0	0	0
EAGen	JZ			0	IFetch	0	0	1	0	0	+	PC	0	0	0
EAGen	JZ			0 1	IFetch NoJump	0	0	1	0	0	+	PC		0	0
EAGen	010X				Opnd	0	0	1	0	0				0	0
EAGen	10XX				Opnd	0	0	1	0	0				0	0
EAGen	0011				?	0	0	0	0	0				0	0
EAGen	010X				?	0	0	0	0	0				0	0

Green is Moore Solution

Control Signal Table (Cont).

Q	IR	s	A_{11}	=0	Q*	LA	LPC	LMAR	LMDR	LIR	ALU	bMUX	aGate	Read	Write
Opnd	LOAD				Execute	0	0	0	1	0				1	0
Opnd	10XX				Execute	0	0	0	1	0				1	0
Opnd	STORE				IFetch	0	0	0	0	0				0	1
Execute	LOAD				IFetch	1	0	0	0	0	+	MDR	0	0	0
Execute	AND				IFetch	1	0	0	0	0	AND	MDR	1	0	0
Execute	OR				IFetch	1	0	0	0	0	OR	MDR	1	0	0
Execute	ADD				IFetch	1	0	0	0	0	+	MDR	1	0	0
Execute	SUB				IFetch	1	0	0	0	0	-	MDR	1	0	0
NoJump					IFetch	0	0	1	0	0	+	PC	0	0	0

NoJump is an added state to avoid Mealy Problem

Designing for a Microprogrammed Implementation

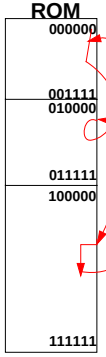
Decisions to make:

- What is overall structure of microprogram in terms of microaddresses
- How are we going to sequence thru these microinstructions
- How do we design the microsequencer
- What are the fields of the microinstructions

There are no *single correct* answers! *Many solutions* possible

What is overall Structure & Sequencing

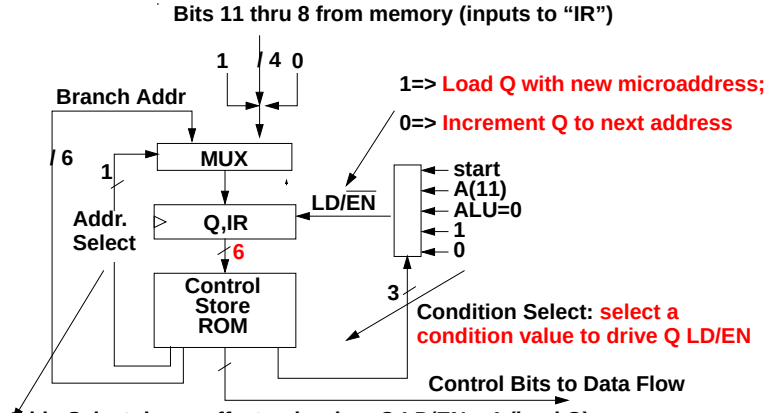
- Some states like Stopped, IFetch are common to all opcodes
- There are 16 opcodes, so it makes sense to at some point have 16 separate microinstructions, one for each opcode
- Each opcode has a sequence of somewhat distinct OperandAccess & Execute microinstructions
- Prior tables identified potentially 22 separate microinstructions: assume we have room for 64 (some "slop" for additions later)



1. Place microinstructions for common states here
   - Conditional branching on start, etc
   - "Case statement" Multiway branch at end
3. Place longer microprograms for each opcode here
   - Sequential execution of individual microinstructions
   - At end, branch back to common routines
2. Place up to 2 microinstructions for each opcode here
   - Normally branch back to IFetch
   - Branch out to separate microprogram, if more space needed

Microprogrammed Design: Sequencing Logic

Note: we will make IR "part" of Q



Addr. Select: has a effect only when Q LD/EN = 1 (load Q)

- 0 => select next address from bits in the ROM (Branch Address)
- 1 => construct address from control store bits & "IR" bits

Microinstruction Template

000: False
001: True
010: ~A(11)
011: ~ALU=0
100: ~start

0: Don't Load Register
1: Load Register

00: b<-0
10: b<-MDR
11: b<-PC

0: a<-0
1: a<-A

Cond Sel	Addr Sel	Next Adr	Load A	Load PC	Load MAR	Load MDR	ALU	b MUX	a Gate	Read	Write
3	1	6	1	1	1	1	4	2	1	1	1

b-invert	carry-in	00: AND 01: OR 10: ADD

0: No Read
1: Mem Read

0: Use Next_Adr
1: 1,MemBus(11:8),0

0: No Write
1: Mem Write

Sample Microinstruction

000: False
001: True
010: ~A(11)
011: ~ALU=0
100: ~start

Stopped: at location 000000, with IFetch at 000001
Note that this microinstruction loops on itself

0: Don't Load Register
1: Load Register

00: b<-0
10: b<-MDR
11: b<-PC

0: a<-0
1: a<-A

Cond Sel	Addr Sel	Next Adr	Load A	Load PC	Load MAR	Load MDR	ALU	b MUX	a Gate	Read	Write
3	1	6	1	1	1	1	4	2	1	1	1
100	0	000000	0	1	1	0	0010	0	0	0	0

Clear PC and test start

b-invert	carry-in	00: AND 01: OR 10: ADD

0: No Read
1: Mem Read

0: Use Next_Adr
1: 1,MemBus(11:8),0

0: No Write
1: Mem Write

Sample Microinstruction

000: False
001: True
010: ~A(11)
011: ~ALU=0
100: ~start

IFetch: at location 000001. Read, PC++

0: Don't Load Register
1: Load Register

00: b<-0
10: b<-MDR
11: b<-PC

0: a<-0
1: a<-A

Cond Sel	Addr Sel	Next Adr	Load A	Load PC	Load MAR	Load MDR	ALU	b MUX	a Gate	Read	Write
3	1	6	1	1	1	1	4	2	1	1	1
001	1		0	1	1	1	0110	11	0	1	0

b-invert	carry-in	00: AND 01: OR 10: ADD
0	1	

0: No Read
1: Mem Read

0: Use Next_Adr
1: 1,MemBus(11:8),0

0: No Write
1: Mem Write

Your Turn

000: False
001: True
010: ~A(11)
011: ~ALU=0
100: ~start

Let's do EAGen for JN (at location 100010)
MAR <-MDR(7:0) || if A(11) then (PC<-MDR(7:0) || ->IFetch)
else -> NoJump

0: Don't Load Register
1: Load Register

00: b<-0
10: b<-MDR
11: b<-PC

0: a<-0
1: a<-A

Cond Sel	Addr Sel	Next Adr	Load A	Load PC	Load MAR	Load MDR	ALU	b MUX	a Gate	Read	Write
3	1	6	1	1	1	1	4	2	1	1	1

b-invert	carry-in	00: AND 01: OR 10: ADD

0: No Read
1: Mem Read

0: Use Next_Adr
1: 1,MemBus(11:8),0

Note: we need a 2nd uword for the conditional PC load

0: No Write
1: Mem Write

Microprogram (page 1)

Adr	Cond Sel	Addr Sel	Next Adr	Load A	Load PC	Load MAR	Load MDR	ALU	b MUX	a Gate	Read	Write
000000	100	0	000000	0	1	1	0	0010	0	0	0	0
000001	001	1		0	1	1	1	0110	11	0	1	0
000010												
000011												
000100												
000101												
000110												
000111												
001000												
001001												
001010												
001011												
001100												
001101												
001110												
001111												

- Stopped = 000000: clear PC & test start
- IFetch=000001: Do a multiway branch by loading IR (part of Q)

Microprogram (page 2)

Adr	Cond Sel	Addr Sel	Next Adr	Load A	Load PC	Load MAR	Load MDR	ALU	b MUX	a Gate	Read	Write
010000												
010001												
010010												
010011												
010100	001	0	000001	1	0	0	0	0010	10	0	0	0
010101												
010110												
010111												
011000	001	0	000001	1	0	0	0	0000	10	1	0	0
011001	001	0	000001	1	0	0	0	0001	10	1	0	0
011010	001	0	000001	1	0	0	0	0010	10	1	0	0
011011	001	0	000001	1	0	0	0	1110	10	1	0	0
011100												
011101												
011110												
011111												

* The microinstructions here are ones that overflowed from the latter ha
* Suggestion: have 01wxyz be a spare location for opcode program 1wxyz0, not all opcodes need an extra spot.

Microprogram (page 3)

Adr	Cond Sel	Addr Sel	Next Adr	Load A	Load PC	Load MAR	Load MDR	ALU	b MUX	a Gate	Read	Write
100000	001	0	000001	0	1	1	0	0010	10	0	0	0
100001												
100010	010	0	000001	0	0	0	0				0	0
100011	001	0	000001	0	1	1	0	0010	10	0	0	0
100100	011	0	000001	0	0	0	0					
100101	001	0	000001	0	1	1	0	0010	10	0		
100110												
100111												
101000	000			0	0	1	0	0010	10	0	0	0
101001	001	0	010100	0	0	0	1				1	0
101010	000			0	0	1	0	0010	10	0	0	0
101011	001	0	000001	0	0	0	0				0	1
101100												
101101												
101110												
101111												

JMP
JN
JZ
LOAD
STORE

- Miccroaddress of 1wxyz0 is 1st of 2 locations set aside for opcode wxyz, starting with EAGen
- Thus 101000 is start of microprogram for LOAD (opcode 0100) with 101001 being the operandaccess cycle

Microprogram (page 4)

Adr	Cond Sel	Addr Sel	Next Adr	Load A	Load PC	Load MAR	Load MDR	ALU	b MUX	a Gate	Read	Write
110000	000			0	0	1	0	0010	10	0	0	0
110001	001	0	011000	0	0	0	1				1	0
110010	000			0	0	1	0	0010	10	0	0	0
110011	001	0	011001	0	0	0	1				1	0
110100	000			0	0	1	0	0010	10	0	0	0
110101	001	0	011010	0	0	0	1				1	0
110110	000			0	0	1	0	0010	10	0	0	0
110111	001	0	011011	0	0	0	1				1	0
111000												
111001												
111010												
111011												
111100												
111101												
111110												
111111												

OR