Systems I: Computer Organization and Architecture

Lecture 10: Microprogrammed Control

Microprogramming

- The control unit is responsible for initiating the sequence of microoperations that comprise instructions.
  - When these control signals are generated by hardware, the control unit is ***hardwired***.
  - When these control signals originate in data stored in a special unit and constitute a program on the small scale, the control unit is ***microprogrammed***.

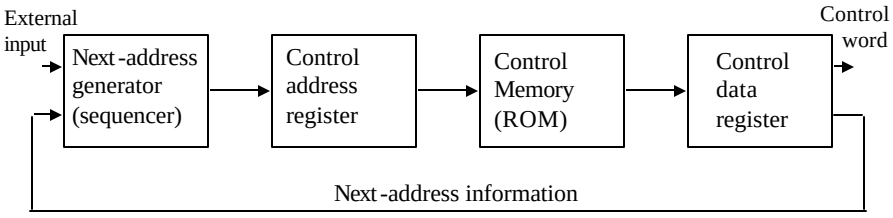
Control memory

- The control function specifying a microoperation is a binary variable whose active state could be either 1 or 0.
  - In the variable's active state, the microoperation is executed.
  - The string of control variables which control the sequence of microoperations is called a ***control word***.
- The microoperations specified in a control word is called a ***microinstruction***.
  - Each microinstruction specifies one or more microoperations that is performed.
- The control unit coordinates stores microinstruction in its own memory (usually ROM) and performed the necessary steps to execute the sequences of microinstructions (called microprograms).

The Microprogrammed Control Unit

- In a microprogrammed processor, the control unit consists of:
  - ***Control address register*** – contains the address of the next microinstruction to be executed.
  - ***Control data register*** – contains the microinstruction to be executed.
  - ***The sequencer*** – determines the next address from within control memory
  - ***Control memory*** – where microinstructions are stored.

Microprogrammed Control Organization



Sequencer

- The sequencer generates a new address by:
  - incrementing the CAR
  - loading the CAR with an address from control memory.
  - transferring an external address

or

  - loading an initial address to start the control operations.

Address Sequencing

- Microinstructions are usually stored in groups where each group specifies a routine, where each routine specifies how to carry out an instruction.
- Each routine must be able to branch to the next routine in the sequence.
- An initial address is loaded into the CAR when power is turned on; this is usually the address of the first microinstruction in the instruction fetch routine.
- Next, the control unit must determine the effective address of the instruction.

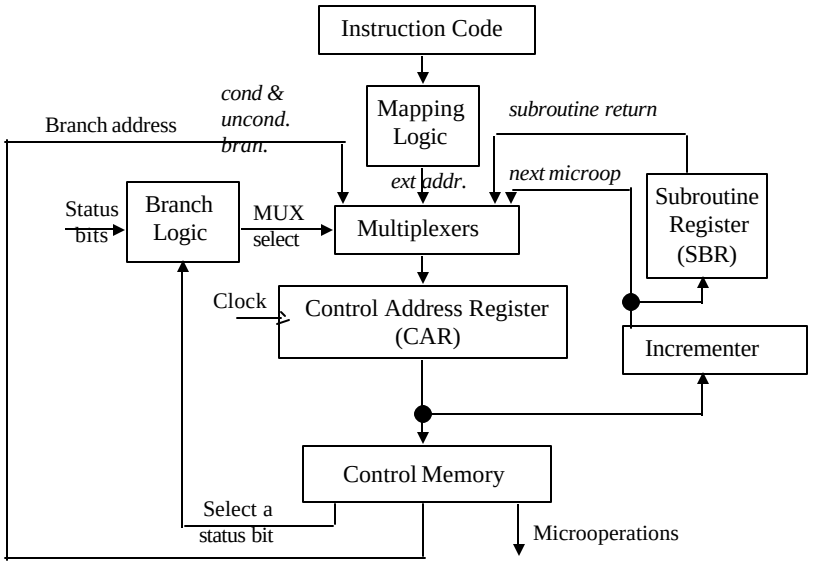
Mapping

- The next step is to generate the microoperations that executed the instruction.
  - This involves taking the instruction's opcode and transforming it into an address for the the instruction's microprogram in control memory. This process is called ***mapping***.
  - While microinstruction sequences are usually determined by incrementing the CAR, this is not always the case. If the processor's control unit can support subroutines in a microprogram, it will need an external register for storing return addresses.

Addressing Sequencing (continued)

- When instruction execution is finished, control must be return to the fetch routine. This is done using an unconditional branch.
- Addressing sequencing capabilities of control memory include:
  - Incrementing the CAR
  - Unconditional and conditional branching (depending on status bit).
  - Mapping instruction bits into control memory addresses
  - Handling subroutine calls and returns.

Selection Of Address For Control Memory



Conditional Branching

- Status bits
  - provide parameter information such as the carry-out from the adder, sign of a number, mode bits of an instruction, etc.
  - control the conditional branch decisions made by the branch logic together with the field in the microinstruction that specifies a branch address.

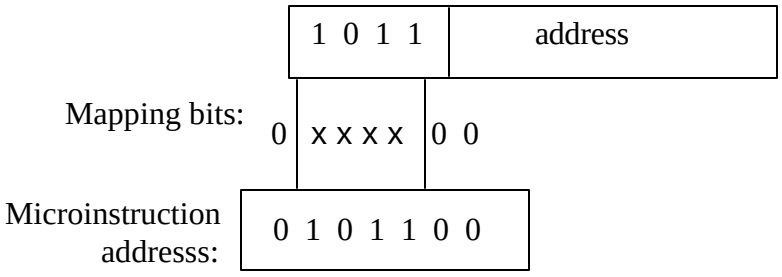
Branch Logic

- Branch Logic - may be implemented in one of several ways:
  - The simplest way is to test the specified condition and branch if the condition is true; else increment the address register.
  - This is implemented using a multiplexer:
    - If the status bit is one of eight status bits, it is indicated by a 3-bit select number.
    - If the select status bit is 1, the output is 0; else it is 0.
    - A 1 generates the control signal for the branch; a 0 generates the signal to increment the CAR.
- Unconditional branching occurs by fixing the status bit as always being 1.

Mapping of Instruction

- Branching to the first word of a microprogram is a special type of branch. The branch is indicated by the opcode of the instruction.
- The mapping scheme shown in the figure allows for four microinstruction as well as overflow space from 1000000 to 1111111.

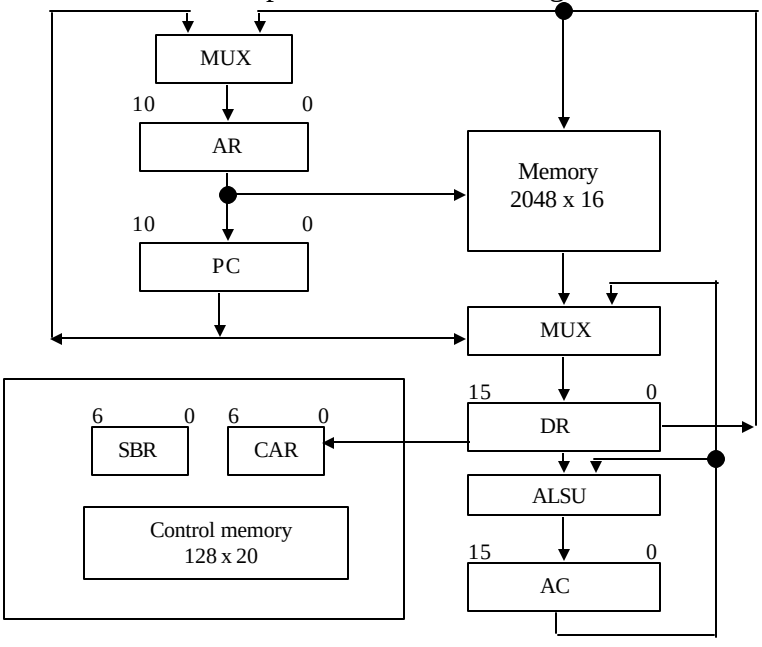
Mapping From Instruction Code To Microoperation Address



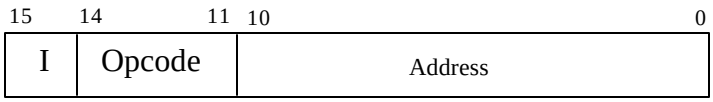
Subroutines

- Subroutine calls are a special type of branch where we return to one instruction below the calling instruction.
  - Provision must be made to save the return address, since it cannot be written into ROM.

Computer Hardware Configuration

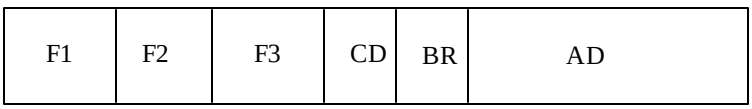


Computer Instructions



Symbol	**Opcode**	**Description**
ADD	0000	AC← AC + M[EA]
BRANCH	0001	IF (AC > 0) THEN PC ← EA
STORE	0010	M[EA] ← AC
EXCHANGE	0011	AC ← M[EA], M[EA]← AC

Microinstruction Code Format (20 bits)



F1, F2, F3 : Microoperation Field

CD: Condition For Branching

BR: Branch Field

AD: Address Field

Symbols and Binary Code For Microinstruction Fields

F1	**Microoperation**	**Symbol**
000	None	NOP
001	AC ← AC + DR	ADD
010	AC ← 0	CLRAC
011	AC ← AC + 1	INCAC
100	AC ← DR	DRTAC
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE

Symbols and Binary Code For Microinstruction Fields (continued)

F2	**Microoperation**	**Symbol**
000	None	NOP
001	AC ← AC- DR	SUB
010	AC ← AC ∨ DR	OR
011	AC ← AC ∧ DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTDR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR

Symbols and Binary Code For Microinstruction Fields (continued)

F3	**Microoperation**	**Symbol**
000	None	NOP
001	AC ← AC ⊕ DR	XOR
010	AC ← AC'	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	Reserved	

Symbols and Binary Code For Microinstruction Fields (continued)

CD	**Condition**	**Symbol**	**Comments**
00	Always = 1	U	Unconditional Branch
01	DR(15)	I	Indirect Address bit
10	AC(15)	S	Sign bit of AC
11	AC = 0	Z	Zero value in AC

Symbols and Binary Code For Microinstruction Fields (continued)

BR	**Symbol**	**Function**
00	JMP	CAR ←AR if condition = 1 CAR←CAR + 1 if condition = 0
01	CAL	CAR ←AR, SBR ← CAR + 1 if cond. = 1 CAR←CAR + 1 if condition = 0
10	RET	CAR ← SBR (return from subroutine)
11	MAP	CAR(2-5) ← DR(11-14), CAR(0, 1, 6) ← 0

Symbolic Microinstructions

- It is possible to create a symbolic language for microcode that is machine-translatable to binary code.
- Each line define a symbolic microinstruction with each column defining one of five fields:
  - **Label** - Either blank or a name followed by a colon (*indicates a potential branch*)
  - **Microoperations** - One, Two, Three Symbols, separated by commas (*indicates that the microoperation being performed*)
  - **CD** - Either U, I, S or Z (*indicates condition*)
  - **BR** - One of four two-bit numbers
  - **AD** - A Symbolic Address, NEXT (address), RET, MAP (both of these last two converted to zeros by the assembler) (*indicates the address of the next microinstruction*)
- We will use the pseudoinstruction ORG to define the first instruction (or origin) of a microprogram, e.g., ORG 64 begins at 1000000.

Partial Symbolic Microprogram

Label	Microoperations	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH

Partial Symbolic MicroProgram (continued)

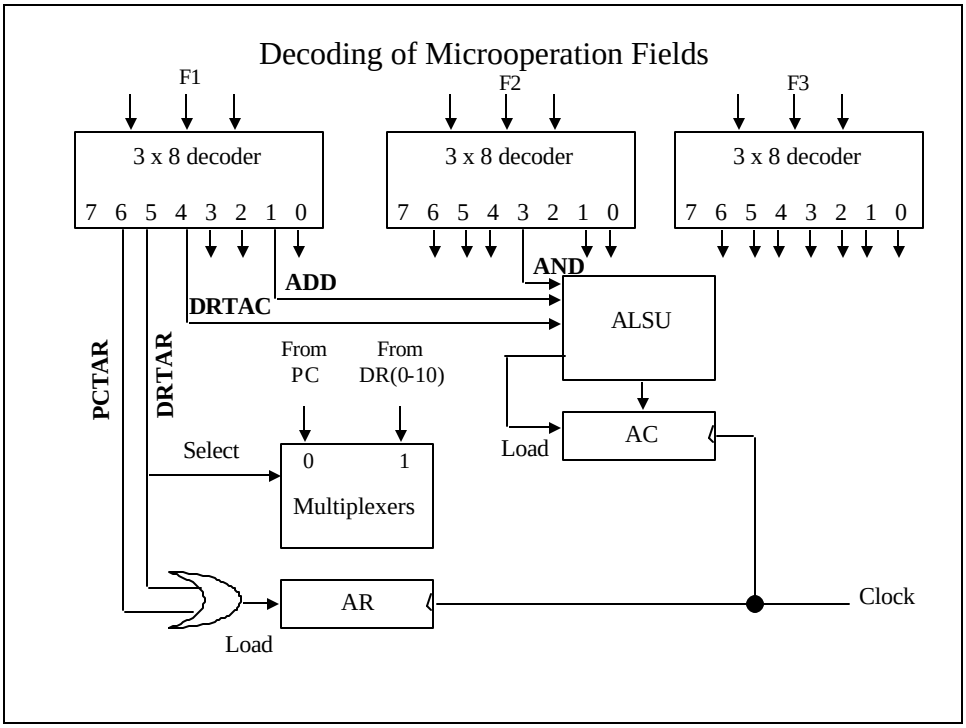
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ARTDR, DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAC	U	MAP	
INDRCT:	READ	U	JMP	NEXT
	DRTAC	U	RET	

Partial Binary Microprogram

	Address		Binary Microinstruction					
Micro-Routine	Decimal	Binary	F1	F2	F3	CD	BR	AD
ADD	0	0000000	000	000	000	01	01	1000011
	1	0000001	000	100	000	00	00	0000010
	2	0000010	001	000	000	00	00	1000000
	3	0000011	000	000	000	00	00	1000000
BRANCH	4	0000100	000	000	000	10	00	0000110
	5	0000101	000	000	000	00	00	1000000
	6	0000110	000	000	000	01	01	1000011
	7	0000111	000	000	110	00	00	1000000
STORE	8	0001000	000	000	000	01	01	1000011
	9	0001001	000	101	000	00	00	0001010
	10	0001010	111	000	000	00	00	1000000
	11	0001011	000	000	000	00	00	1000000
EXCHANGE	12	0001100	000	000	000	01	01	1000011
	13	0001101	001	000	000	00	00	0001110
	14	0001110	100	101	000	00	00	0001111
	15	0001111	111	000	000	00	00	1000000
FETCH	64	1000000	000	000	000	00	00	1000001
	65	1000001	000	100	000	00	00	1000010
	66	1000010	000	000	000	00	11	0000000
INDRCT	67	1000011	000	100	000	00	00	1000100
	68	1000100	000	000	000	00	10	0000000

Control Unit Design

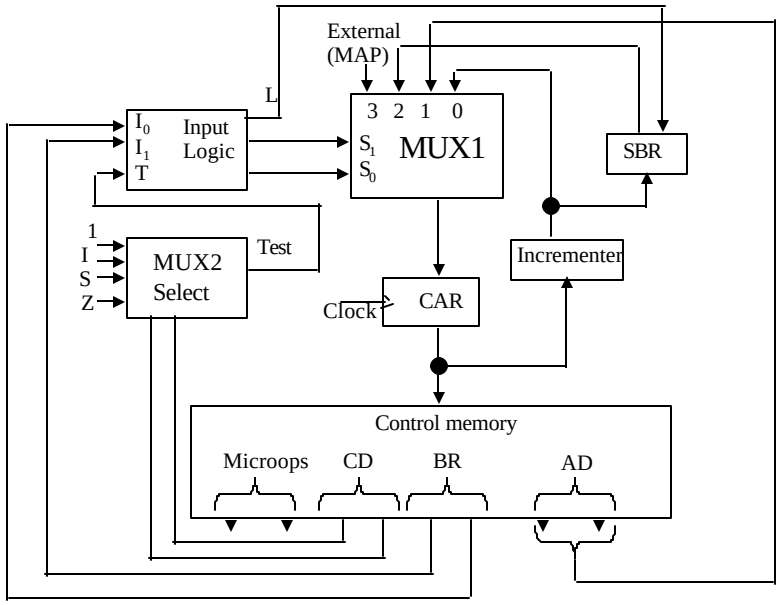
- Each field of k bits allows for 2^k microoperations.
- The number of control bits can be reduced by grouping mutually exclusive microoperations together.
- Each field requires its own decoder to produce the necessary control signals.



Microprogram Sequencer

- The microprogram sequencer selects the next address in control memory from which a microinstruction is to be fetched.
- Depending on the condition and on the branching type, it will be:
  - an external (mapped) address
  - the next microinstruction
  - a return from a subroutine
  - the address indicated in the microinstruction.

Microprogram Sequencer For A Control Memory



Input Logic Truth Table For A Microprogrammed Sequencer

BR Field		**Input**			**MUX 1**		**Load SBR**	
		I_1	**I_0**	**T**	**S_1**	**S_0**	**L**	
0	0	0	0	0	0	0	0	Next address
0	0	0	0	1	0	1	0	Specified addr.
0	1	0	1	0	0	0	0	
0	1	0	1	1	0	1	1	
1	0	1	0	x	1	0	0	Subroutine ret.
1	1	1	1	x	1	1	0	Ext. addr.