ECE 3401 Lecture 20

Microprogramming (III)

Overview

- Part 1 – Datapaths
  - Introduction
  - Datapath Example
  - Datapath Representation and Control Word
- Part 2 – A Simple Computer
  - Instruction Set Architecture (ISA)
  - Single-Cycle Hardwired Control
    - PC Function
    - Instruction Decoder
    - Example Instruction Execution
- Part 3 – Multiple Cycle Hardwired Control
  - Single Cycle Computer Issues
  - Sequential Control Design

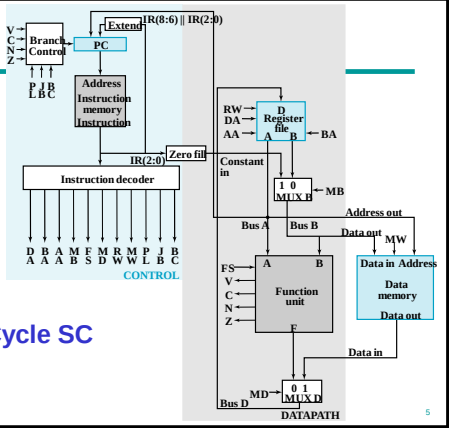
Single-Cycle Computer Issues

- Shortcoming of Single Cycle Design
  - Complexity of instructions executable in a single cycle is limited
  - Accessing both an instruction and data from a simple single memory impossible
  - A long worst case delay path limits clock frequency and the rate of performing instructions
- Handling of Shortcomings
  - The first two shortcomings can be handled by the multiple-cycle computer
  - The third shortcoming is dealt with by using a technique called pipelining described in later lectures

Multiple-Cycle Computer

- Converting the single-cycle computer into a multiple-cycle computer involves:
  - Modifications to the datapath/memory
  - Modification to the control unit
  - Design of a multiple-cycle hardwired control

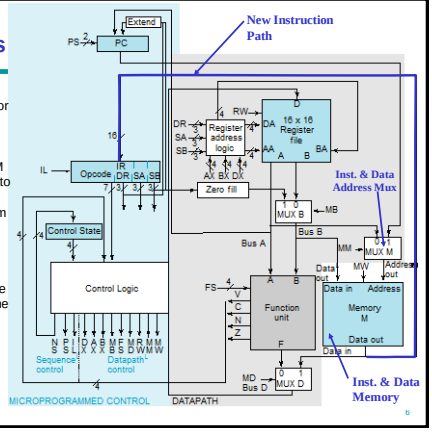
Single-Cycle SC



Datapath Modifications

Use a single memory for both instructions and data

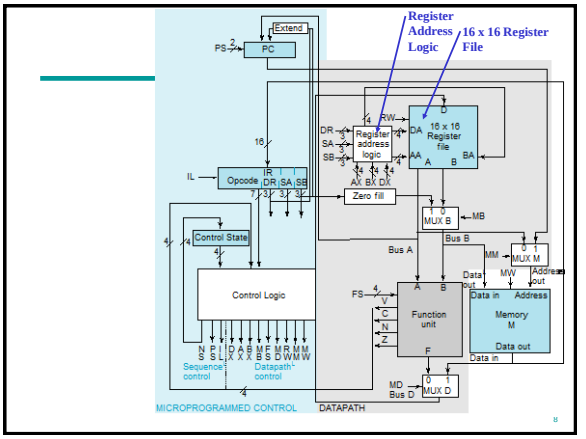
- Requires new MUX M with control signal MM to select between the instruction address from the PC and the data address
- Requires path from Memory Data Out to the instruction register in the control unit



Datapath Modifications (Continued)

- Additional registers needed to hold operands between cycles
  - Add 8 temporary storage registers to the Register File
    - Register File becomes 16 x 16
    - Addresses to Register File increase from 3 to 4 bits
  - Register File addresses come from:
    - The instruction for the Storage Resource registers (0 to 7)
    - The control word for the Temporary Storage registers (8 to 15)
  - Add Register Address Logic to the Register File to select the register address sources
    - Three new control fields for register address source selection and temporary storage addressing: DX, AX, BX

7

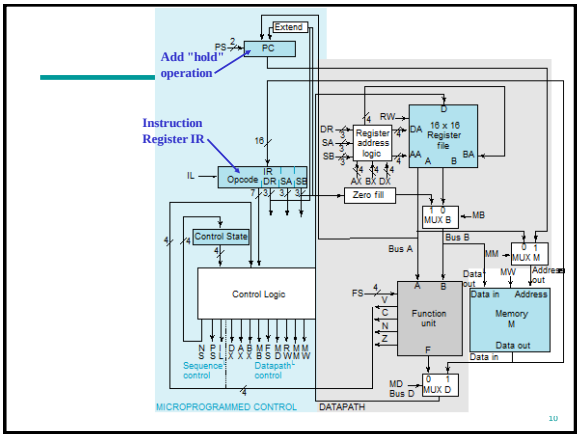


8

Control Unit Modifications

- Must hold instruction over the multiple cycles to draw on instruction information throughout instruction execution
  - Requires an Instruction Register (IR) to hold the instruction
    - Load control signal IL
  - Requires the addition of a "hold" operation to the PC since it only counts up to obtain a new instruction
    - New encoding for the PC operations uses 2 bits

9

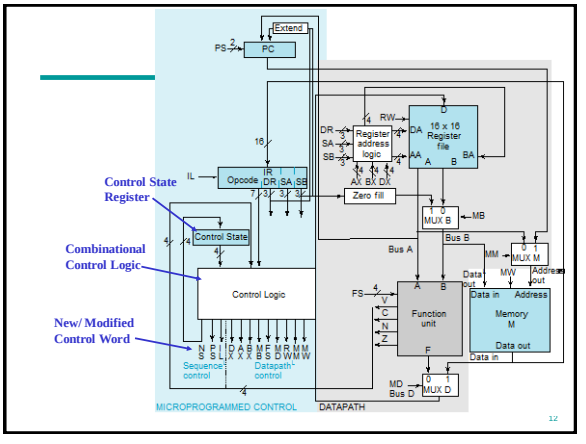


10

Sequential Control Design

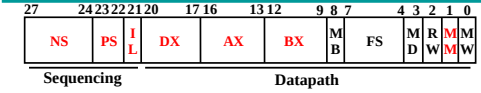
- To control microoperations over multiple cycles, a Sequential Control replaces the Instruction Decoder
  - Input: Opcode, Status Bits, Control State
  - Output:
    - Control Word (Modified Datapath Control part)
    - Next State: Control Word (New Sequencing Control part)
  - Consists of:
    - Register to store the Control State
    - Combinational Logic to generate the Control Word (both sequencing and datapath control parts)
  - The Combinational Logic is quite complex so we assume that it is implemented by using a PLA or synthesized logic and focus on ASM level design

11



12

Control Word



- Datapath part: field MM added, and fields DX, AX, and BX replace DA, AA, and BA, respectively
  - If the MSB of a field is 0, e.g., AX = 0XXX, then AA is 0 concatenated with SA (3bits) field in the IR
  - If the MSB of a field is 1, e. g. AX = 1011, then AA = 1011
- Sequencing part:
  - IL controls the loading of the IR
  - PS controls the operations of the PC
  - NS gives the next state of the Control State register
    - E.g., NS is 4 bits, the length of the Control State register - 16 states are viewed as adequate for this design

Encoding for Datapath Control

DX	AX	BX	Code	MB	Code	FS	Code	MD	RW	MM	MW	Code
R[DR]	*R*[SA]	*R*[SB]	0XXX	Register	0	$F \leftarrow A$	0000	FnUt	No write	Address Out	No write	0
*R*8	*R*8	*R*8	1000	Constant	1	$F \leftarrow A + 1$	0001	Data In	Write	PC	Write	1
*R*9	*R*9	*R*9	1001			$F \leftarrow A + B$	0010					
*R*10	*R*10	*R*10	1010			Unused	0011					
*R*11	*R*11	*R*11	1011			Unused	0100					
*R*12	*R*12	*R*12	1100			$F \leftarrow A + \overline{B} + 1$	0101					
*R*13	*R*13	*R*13	1101			$F \leftarrow A - 1$	0110					
*R*14	*R*14	*R*14	1110			Unused	0111					
*R*15	*R*15	*R*15	1111			$F \leftarrow A \wedge B$	1000					
						$F \leftarrow A \vee B$	1001					
						$F \leftarrow A \oplus B$	1010					
						$F \leftarrow \overline{A}$	1011					
						$F \leftarrow B$	1100					
						$F \leftarrow$ sr B	1101					
						$F \leftarrow$ sl B	1110					
						Unused	1111					

Encoding for Sequencing Control

NS	PS		IL	
Next State	Action	Code	Action	Code
Gives next state of Control State Register	Hold PC Inc PC Branch Jump	00 01 10 11	No load Load instr.	0 1

ASM Charts for Sequential Control

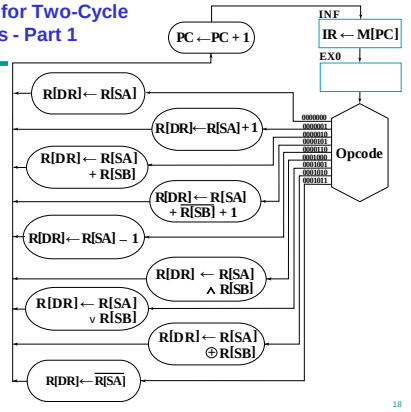
- An instruction requires two steps:
  - *Instruction fetch* – obtaining an instruction from memory
  - *Instruction execution* – the execution of a sequence of microoperations to perform instruction processing
  - Due to the use of the IR, these two steps require a minimum of two clock cycles
- ISA: Instruction Specifications and ASM charts for the instructions (that all require two clock cycles)
  - A vector decision box is used for the opcode
  - Scalar decision boxes are used for the status bits

ISA: Instruction Specifications (for reference)

Instruction Specifications for the Simple Computer - Part 1

Instruction	Opcode	Mnemonic	Format	Description	Status Bits
Move A	0000000	MOVA	RD,RA	R[DR] ← R[SA]	N, Z
Increment	0000001	INC	RD,RA	R[DR] ← R[SA] + 1	N, Z
Add	0000010	ADD	RD,RA,RB	R[DR] ← R[SA] + R[SB]	N, Z
Subtract	0000101	SUB	RD,RA,RB	R[DR] ← R[SA] − R[SB]	N, Z
Decrement	0000110	DEC	RD,RA	R[DR] ← R[SA] − 1	N, Z
AND	0001000	AND	RD,RA,RB	R[DR] ← R[SA] ∧ R[SB]	N, Z
OR	0001001	OR	RD,RA,RB	R[DR] ← R[SA] ∨ R[SB]	N, Z
Exclusive OR	0001010	XOR	RD,RA,RB	R[DR] ← R[SA] ⊕ R[SB]	N, Z
NOT	0001011	NOT	RD,RA	R[DR] ← $\overline{\text{R[SA]}}$	N, Z

ASM Chart for Two-Cycle Instructions - Part 1



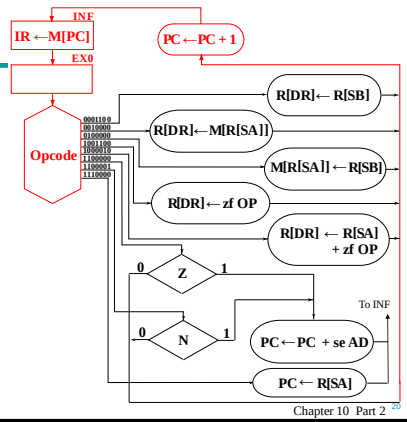
ISA: Instruction Specifications (for reference)

Instruction Specifications for the Simple Computer - Part 2

Instruction	Opcode	Mnemonic	Format	Description	Status Bits
Move B	0001100	MOVB	RD,RB	R[DR] ← R[SB]	
Shift Right	0001101	SHR	RD,RB	R[DR] ← sr R[SB]	
Shift Left	0001110	SHL	RD,RB	R[DR] ← sl R[SB]	
Load Immediate	1001100	LDI	RD, OP	R[DR] ← zf OP	
Add Immediate	1000010	ADI	RD,RA,OP	R[DR] ← R[SA] + zf OP	
Load	0010000	LD	RD,RA	R[DR] ← M[SA]	
Store	0100000	ST	RA,RB	M[SA] ← R[SB]	
Branch on Zero	1100000	BRZ	RA,AD	if (R[SA] = 0) PC ← PC + se A D	
Branch on Negative	1100001	BRN	RA,AD	if (R[SA] < 0) PC ← PC + se A D	
Jump	1110000	JMP	RA	PC ← R[SA]	

ASM Chart for 2-Cycle Instructions - Part 2

- Portion in Red duplicated from previous ASM chart



State Table for 2-Cycle Instructions

State	Inputs: Opcode	Inputs: VCNZ	Next state	Outputs: IL	PS	DX	AX	BX	MB	FS	MD	RW	MM	MW	Comments
INF	XXXXXXX	XXXX	EX0	1	00	XXXX	XXXX	XXXX	X	XXXX	X	0	1	0	IR ← M[PC]
EX0	0000000	XXXX	INF	0	01	0XXX	0XXX	XXXX	X	0000	0	1	X	0	R[DR] ← R[SA]*
EX0	0000001	XXXX	INF	0	01	0XXX	0XXX	XXXX	X	0001	0	1	X	0	INC R[DR] ← R[SA] + 1*
EX0	0000010	XXXX	INF	0	01	0XXX	0XXX	XXXX	0	0010	0	1	X	0	ADD R[DR] ← R[SA] + R[SB]*
EX0	0000101	XXXX	INF	0	01	0XXX	0XXX	0XXX	0	0101	0	1	X	0	SUB R[DR] ← R[SA] + $\overline{R[SB]}$ + 1*
EX0	0000110	XXXX	INF	0	01	0XXX	0XXX	XXXX	X	0110	0	1	X	0	DEC R[DR] ← R[SA] + (- 1)*
EX0	0001000	XXXX	INF	0	01	0XXX	0XXX	0XXX	0	1000	0	1	X	0	AND R[DR] ← R[SA] ∧ R[SB]*
EX0	0001001	XXXX	INF	0	01	0XXX	0XXX	0XXX	0	1001	0	1	X	0	OR R[DR] ← R[SA] ∨ R[SB]*
EX0	0001010	XXXX	INF	0	01	0XXX	0XXX	0XXX	0	1010	0	1	X	0	XOR R[DR] ← R[SA] ⊙ R[SB]*
EX0	0001011	XXXX	INF	0	01	0XXX	0XXX	XXXX	X	1011	0	1	X	0	NOT R[DR] ← $\overline{R[SA]}$ *
EX0	0001100	XXXX	INF	0	01	0XXX	XXXX	0XXX	0	1100	0	1	X	0	MOVB R[DR] ← R[SB]*
EX0	0010000	XXXX	INF	0	01	0XXX	0XXX	XXXX	X	XXXX	1	1	0	0	LD R[DR] ← M[R[SA]]*
EX0	0100000	XXXX	INF	0	01	XXXX	0XXX	0XXX	0	XXXX	X	0	0	1	ST M[R[SA]] ← R[SB]*
EX0	1001100	XXXX	INF	0	01	0XXX	XXXX	XXXX	1	1100	0	1	0	0	LDI R[DR] ← zf OP*
EX0	1000010	XXXX	INF	0	01	0XXX	0XXX	XXXX	1	0010	0	1	0	0	ADI R[DR] ← R[SA] + zf OP*
EX0	1100000	XXX1	INF	0	10	XXXX	0XXX	XXXX	X	0000	X	0	0	0	BRZ PC ← PC + se AD
EX0	1100000	XXX0	INF	0	01	XXXX	0XXX	XXXX	X	0000	X	0	0	0	BRZ PC ← PC + 1
EX0	1100001	XX1X	INF	0	10	XXXX	0XXX	XXXX	X	0000	X	0	0	0	BRN PC ← PC + se AD
EX0	1100001	XX0X	INF	0	01	XXXX	0XXX	XXXX	X	0000	X	0	0	0	BRN PC ← PC + 1
EX0	1110000	XXXX	INF	0	11	XXXX	0XXX	XXXX	X	0000	X	0	0	0	JMP PC ← R[SA]

* For this state and input combinations, PC ← PC+1 also occurs

3-Process ASM VHDL Code

```
entity controller is
   Port ( opcode : in std_logic_vector(6 downto 0);
          reset, clk : in std_logic;
          zero, negative : in std_logic;
          IL, MB, MD, MM, RW, MW : out std_logic;
          PS : out std_logic_vector(1 downto 0);
          DX, AX, BX, FS : out std_logic_vector(3 downto 0);
);
end controller;

architecture Behavioral of controller is
  type state_type is (RES, FTH, EX);
  signal cur_state, next_state : state_type;
begin
        state_register:process(clk, reset)
        begin
                if(reset='1') then
                        cur_state<=RES;
                elsif (clk'event and clk='1') then
                        cur_state<=next_state;
                end if;
        end process;
```


3-Process ASM VHDL Code

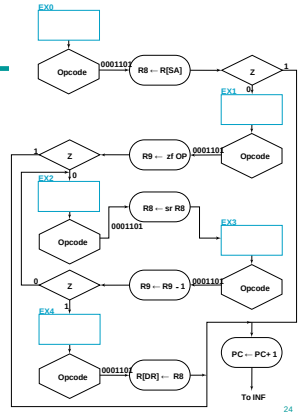
```
out_func: process (cur_state, opcode, zero, negative)
  begin
        (IL,PS, MB, FS, MD, RW, MW, MM) <= std_logic_vector'(0x"000");
        FS<="0000";
          case cur_state is
                when RES =>
                  next_state <= FTH;
                when FTH =>
                  -- set the control vector values
                  next_state <= EXE;
                when EXE =>
                        case opcde is
                            when "0000000" =>


end process;

End Behavioral;
```


ASM Chart for Multiple Bits Right Shift

- R8 – used to perform shifts
- R9 – used to store and decrement shift count
- Zero test in EX1 is to determine if the shift amount is 0; if so, goes to state INF



State Table For Multiple Bits Right Shift

State	Inputs		Next state	Outputs											Comments
	Opcode	VCNZ		IL	PS	DX	AX	BX	MB	FS	MD	RW	MM	MW	
EX0	0001101	XXX0	EX1	0	00	1000	0XXX	XXXX	X	0000	0	1	X	0	SRM	R8←R[SA],$\overline{Z}$:→EX1
EX0	0001101	XXX1	INF	0	01	1000	0XXX	XXXX	X	0000	0	1	X	0	SRM	R8←R[SA],Z:→INF*
EX1	0001101	XXX0	EX2	0	00	1001	XXXX	XXXX	1	1100	0	1	X	0	SRM	R9← zf OP,$\overline{Z}$:→EX2
EX1	0001101	XXX1	INF	0	01	1001	XXXX	XXXX	1	1100	0	1	X	0	SRM	R9← zf OP,Z:→INF*
EX2	0001101	XXXX	EX3	0	00	1000	XXXX	1000	0	1101	0	1	X	0	SRM	R8 ← sr R8,→EX3
EX3	0001101	XXX0	EX2	0	00	1001	1001	XXXX	X	0110	0	1	X	0	SRM	R9 ← R9 - 1,$\overline{Z}$:→EX2
EX3	0001101	XXX1	EX4	0	00	1001	1001	XXXX	X	0110	0	1	X	0	SRM	R9 ← R9 - 1,Z:→EX4
EX4	0001101	XXXX	INF	0	01	0XXX	1000	XXXX	X	0000	0	1	X	0	SRM	R[DR] ← R8,→ INF*

* For this state and input combinations, PC ← PC+1 also occurs