EE 231 Lab 8

Computer Control Unit

You are on your way to designing your first computer, but first you need build the control unit. A conceptual block diagram of a simple computer is shown in Figure 1. In previous labs you have already designed the DATA MUX, the ALU, and required registers. The control unit is a finite state machine. Its inputs are the instruction register and the carry as well as a clock pulse and RESET. The control unit's outputs are the control signals that direct the operation of the rest of the computer. The control unit can be in one of four states: RESET, FETCH, EX1 and EX2:

- RESET is the reset cycle. The computer gets into this state when the RESET input is low and stays in this state until the RESET input goes high.

- FETCH is the fetch cycle. The computer program is stored in memory. During the fetch cycle the next instruction is fetched from memory and loaded into the instruction register (INST).

- EX1 is the first execution cycle. Once an instruction has been loaded into INST, the control unit determines the required course of action to take based on the value of INST and the current state of the control unit.

- EX2 is the second execution cycle. Some instructions only require one execution cycle (EX1) while others require two (EX1, and EX2).

1.**Lab**

1.1.Assign op codes to each instruction in the instruction set.

- To improve readability, use **parameter** to assign values that are frequently used in your program, e.g., op codes.

- You should also provide default values for the control signals.

1.2.Write a Verilog program to implement the control unit.

1.3.Simulate the control unit in Altera. **What happens when RESET is low?** Test with different values for INST and check that the control unit cycles through the appropriate states for that instruction and that the control signals are what you expect. **Test the JCS command when the carry is set and when the carry is not set.**

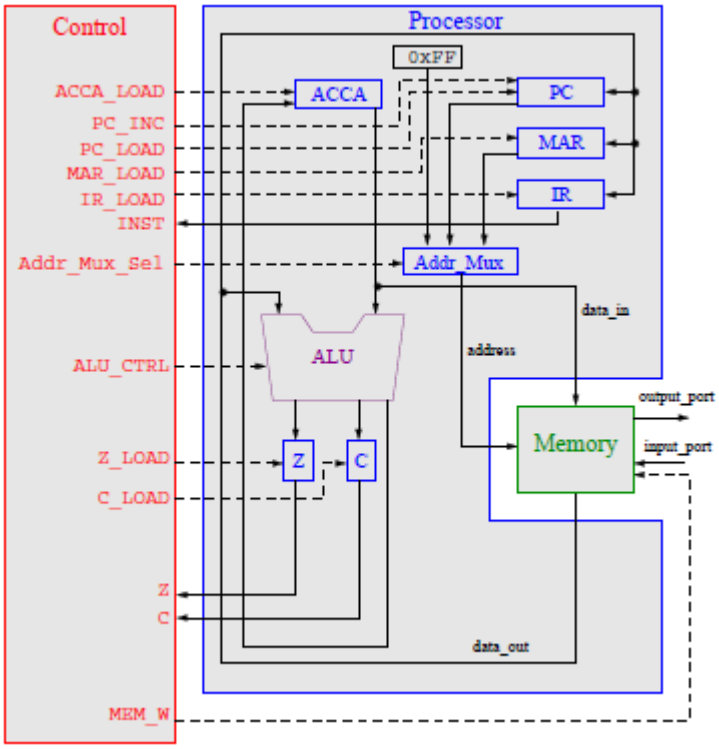

Figure 1: Simple Computer

	Mnemonic	Operation
0	`LDAA addr` (load ACCA from memory)	Loads ACCA with the value in memory at address addr. C stays the same, Z changes
1	`LDAA_IMM #num` (load ACCA with an immediate)	Loads ACCA with num, the value in memory at the address immediately following the LDAA #num command. C stays the same, Z changes
2	`STAA addr` (store ACCA in memory)	Stores the value in ACCA at memory address addr. C stays the same, Z changes
3	`ADDA addr` (add ACCA and value in memory)	Adds the value in memory location addr to the value in ACCA and saves the result inACCA. C and Z change
4	`SUBA addr` (subtract value in memory from ACCA)	Subtracts the value in memory location addr from the value in ACCA and saves the result in ACCA. C and Z change
5	`ANDA addr` (logical AND of ACCA and value in memory)	Perform a logical AND of the value in memory location addr with the value in ACCA. Save the result in ACCA. C stays the same, Z changes
6	`ORAA addr` (logical OR of ACCA and value in memory)	Perform a logical OR of the value in memory location addr with the value in ACCA. Save the result in ACCA. C stays the same, Z changes
7	`CMPA addr` (Compares ACCA to the value in addr)	Compare ACCA to value in addr. This is done by subtracting the value in addr from ACCA. ACCA does not change. C and Z change
8	`COMA` (Complement ACCA)	Replace the value in ACCA with its one's complement. C is set to 1 and Z changes
9	`INCA` (INCA ACCA)	Increment value in ACCA. C stays the same and Z changes
A	`LSLA` (logical shift left ACCA)	Logical shift left of ACCA. C and Z change
B	`LSRA` (logical shift right ACCA)	Logical shift right of ACCA. C and Z change

C	`ASRA` (Arithmetic shift right `ACCA`)	Arithmetic shift right of `ACCA`. `C` and `Z` change
D	`JMP addr` (jump)	Jumps to the instruction stored in address `addr`. The `PC` is replaced with `addr`. `C` and `Z` stay the
E	`JCS addr` (jump if carry set)	Jumps to the instruction stored in address `addr` if `C`=1. If `C` is not set, continue with next instruction. `C` and `Z` stay the same
F	`JCC addr` (jump if carry not set)	Jumps to the instruction stored in address `addr` if `C`=0. If `C` is set, continue with next instruction. `C` and `Z` stay the same
10	`JEQ addr` (jump if `Z` set)	Jumps to the instruction stored in address `addr` if `Z`=1. If `Z` is not set, continue with next instruction. `C` and `Z` stay the same

Table 1: Computer Instructions.

The control unit's outputs are the control signals shown on the block diagram of the computer. Except for ALU_CTRL and MEM_SEL, all of these signals are active low. In your Verilog code you will activate the appropriate signals at the correct times to run the instruction the control unit is executing.

During the FETCH cycle the control unit will fetch the next instruction from memory to determine what instruction it should execute. Thus, the FETCH cycle will be the same for all instructions where it will read the instruction from memory and latch it into the INST register. To do this, IR_LOAD and PC_INC should be low, and MEM_SEL should be set to select the address from the program counter (PC). With the control lines set up this way the address to the memory will be from the PC, which is the address of the next instruction to execute, and the memory output enable line will be low (active). The memory will put the data at that address on its output lines, which are the input lines to the INST register. On the next clock edge, the data from memory will be latched into the INST register, and the PC will increment to the next memory address. What the control unit does next will depend on the data loaded into the INST register. Here are three sample codes of how you may structure your module.

Example 1

Consider the instruction LDAA addr where addr=0×F5. We will from here on assume that the instruction is in memory address 0 × 80 and 0 × 81, and that the code for LDAA addr is 0 × 01.

PC	Memory Address	Memory Data
→	80	01
	81	F5
	82	Next instruction

INST = ??
MAR = ??

FETCH: During the fetch cycle the instruction register must be loaded with the instruction op code, 0 × 01. To do this the Addr MUX Sel must select the PC as the address source and the memory address 0×80 must be read, which causes its value to be placed on the DATA lines. The value on the DATA lines must be latched into IR, and the PC must be incremented. Thus during FETCH you should have PC_INC, INST_LOAD and Addr_Mux_Sel.

PC	Memory Address	Memory Data
	80	01
→	81	F5
	82	Next instruction

INST = 01 (LDAA addr op code)
MAR = ??

EX1: During EX1, you must read the memory address that the PC is pointing at. By reading address 0×81 the value 0×F5 is placed on the DATA line. Then 0×F5 needs to be stored in the MAR register. Finally, the program counter should be incremented. Thus during EX1 you should have PC_INC and MAR_LOAD active, and Addr_Mux_SEL set to PC. After these steps the situation should be as shown below

PC	Memory Address	Memory Data
	80	01
	81	F5
→	02	Next instruction

INST = 01 (LDAA addr op code)
MAR = F5

EX2: Now that MAR contains the value 0×F5, the multiplexer should select MAR as the source of the address. This address should then be read which causes the memory contents of address 0×F5 to be placed onto the DATA line. Then the ALU can load this value into ACCA. During EX2 you should have ACCA_LOAD active, Addr_Mux_SEL set to MAR, and ALU_CTL set to LOAD. When the control lines are set up like this, the value of

0×F5 will be on the address lines of the memory unit, and the data lines out of the memory will contain the data in address 0×F5. This data will be passed through the ALU to the input of ACCA.

On the next clock cycle, the value will be latched into ACCA. Note that you do not want PC_INC active because PC is already pointing to the next instruction to be executed.

Example 2

The next instruction in the program is LDAA #num where #num=0×F5. This instruction translates as "load accumulator ACCA with the value F5". Assume the op code for LDAA # is 0×02. Before the program begins, the situation is as below:

PC	Memory Address	Memory Data
→	82	02
	83	F5
	84	Next instruction

INST = ??
MAR = ??

FETCH: The fetch cycle is the same for this command as it was in Example 1. After the fetch cycle the situation should be:

PC	Memory Address	Memory Data
	82	02
→	83	F5
	84	Next instruction

INST = 02 (LDAA #num op code)
MAR = ??

EX1: During the EX1 cycle the PC is pointing at memory address 0×83. By reading this address, the value 0×F5 is placed on the DATA line. ACCA_LOAD and PC_INC, should be active, MEM SEL should be set to select PC, and the ALU_CTRL lines should select the function which loads ACCA. When the control lines are set up like this, the value 0×83 will be on the address lines of the memory unit, and the data lines out of the memory unit will contain the data in address 0×83 (which in this example is 0×F5). This data will be passed through the ALU to the input of ACCA. On the next clock cycle the data will be latched into ACCA. There is no EX2 cycle.

Example 3

The next instruction in the program is JMP addr where addr=0×F5. Assume the op code for JMP addr is 0×12. Before the program begins, the situation is as below:

PC	Memory Address	Memory Data
→	84	12
	85	F5
	86	Next instruction

```
INST = ??
MAR  = ??
```

FETCH: The fetch cycle is the same for this command as it was in Example 1. After the fetch cycle the situation should be:

PC	Memory Address	Memory Data
	84	12
→	85	F5
	86	Next instruction

```
INST = 12 (JMP addr op code)
MAR  = ??
```

EX1: During the EX1 cycle the PC is pointing at memory address 0×05. By reading this address, the value 0×F5 is placed on the DATA line. ACCA_LOAD and PC_LOAD should be active and MEM_SEL should be set to select PC. When the control lines are set up like this, the value 0×85 will be on the address lines of the memory unit, and the data lines out of the memory unit will contain the data in address 0×85 (which in this example is 0×F5). This data will be on the input lines to PC. On the next clock cycle the data will be latched into PC. There is no EX2 cycle.