Lab 9: Computer Control Unit

October 26, 2009

Contents

1 Prelab

2 Lab

You are on the path do design your first computer. 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. In this lab you will design the computer control unit. 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 state. 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).

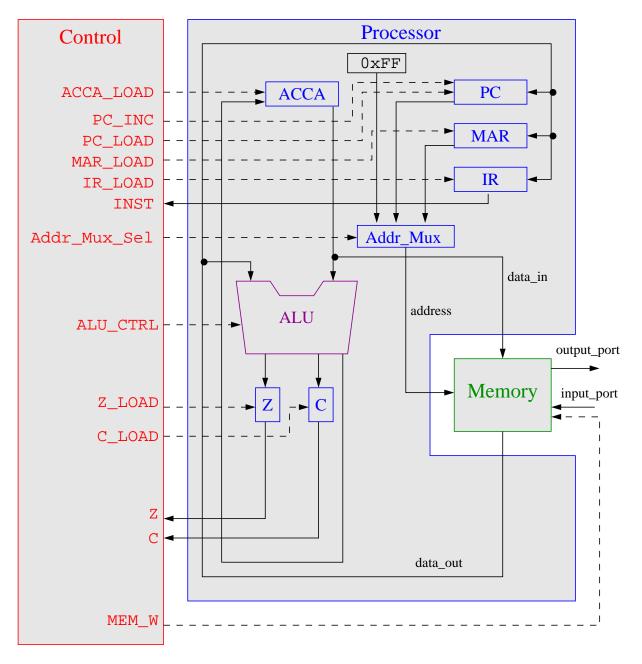


Figure 1: Simple Computer

Table 1:	Computer	Instructions
----------	----------	--------------

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. TMW #num (load ACCA with an inmediate) 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 OR 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 at value in memory) Compare ACCA to value in addr. This is done by subtracting the value in addr from ACCA. ACCA does not change. C and Z changes 9 INCA (LNCA ACCA) Increment value in ACCA. C and Z changes 9 INCA (logical shift right ACCA) Logical shift left of ACCA. C and Z change 9 INCA (logical shift right ACCA) Logical shift right of ACCA. C and Z change 9 JNMP add		Mnemonic	Operation
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) Compare ACCA to value in addr. This is done by subtracting 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 ACCA. C and Z change 8 COMA (logical shift right ACCA) Increment value in ACCA. C and Z changes 9 INCA (logical shift right ACCA) Logical shift right of ACCA. C and Z change 6 BLSA (logical shift right ACCA) Logical shift right of ACCA. C and Z change 9 INCA (logical shift right ACCA) Logical shift ri	0	LDAA addr	Loads ACCA with the value in memory at address addr. C stays the
(load ACCA with an immediate) 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 a saves the result in ACCA. 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 COM (INCA ACCA) Increment value in ACCA. C and Z change 9 INCA (logical shift right ACCA) Logical shift left of ACCA. C and Z change 8 LSLA (logical shift right ACCA) Logical shift right of ACCA. C and Z change 9 INCA (logical shift right ACCA) Jumps to the instruction stored in		(load ACCA from memory)	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 (INCA ACCA) Increment value in ACCA. C and Z change 9 INCA (logical shift left ACCA) Logical shift left of ACCA. C and Z change 8 LSRA (logical shift right ACCA) Logical shift right of ACCA. C and Z change 9 INCA (logical shift right ACCA) Logical shift right of ACCA. C and Z change 10 JMP addr (logical shift right ACCA) Arithmetic shift right of ACCA. C and Z cha	1	LDAA_IMM #num	Loads ACCA with num, the value in memory at the address
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) 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 (INCCA ACCA) Replace the value in ACCA. CCA does not change. C and Z changes 9 INCA (logical shift left ACCA) Logical shift left of ACCA. C and Z change 1 LSLA (logical shift right ACCA) Logical shift right of ACCA. C and Z change 1 Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the same 1 Jumps to the instruction stored in address addr. The PC is replaced wi		(load ACCA with an immediate)	immediately following the LDAA #num command. C stays the same,
(store ACCA in memory) 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 9 INCA (INCA ACCA) Increment value in ACCA. C stays the same and Z changes 9 INCA (logical shift left ACCA) Logical shift left of ACCA. C and Z change 10 JMP addr (logical shift right ACCA) Arithmetic shift right of ACCA. C and Z stay the same 11 Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the same 11 Jumps to the instruction stored in address addr if C=0. If C is set, (jump) if carry set) Jumps to the instruction. C and Z s			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 (INCA (INCA ACCA) Increment value in ACCA. C stays the same and Z changes 9 INCA (logical shift left ACCA) Logical shift left of ACCA. C and Z change 18 LSIA (logical shift right ACCA) Logical shift right of ACCA. C and Z change 19 INCA (logical shift right ACCA) Logical shift right of ACCA. C and Z change 10 JMP addr (logical shift right ACCA) Logical shift right of ACCA. C and Z change 10 JMP addr (lump) Jumps to the instruction stored in address addr. The PC is replaced	2	STAA addr	Stores the value in ACCA at memory address addr. C stays the
(add ACCA and value in memory) 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. C with its one's complement. C is set to 1 and Z changes 9 INCA (logical shift left ACCA) Increment value in ACCA. C and Z change 8 LSLA (logical shift right ACCA) Logical shift right of ACCA. C and Z change 9 INCA (logical shift right ACCA) Arithmetic shift right of ACCA. C and Z change 9 JMP addr (logical shift right ACCA) Logical shift right of ACCA. C and Z change 0 JMP addr (logical shift right ACCA) Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the same		(store ACCA in memory)	same, Z changes
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 ACCA. ACCA does not change. C and Z change 8 COMA (INCCA ACCA) Replace the value in ACCA. C stays the same and Z changes 9 INCA (logical shift left ACCA) Logical shift left of ACCA. C and Z change 8 LSLA (logical shift right ACCA) Logical shift right of ACCA. C and Z change 9 INCA (Arithmetic shift right ACCA) Logical shift right of ACCA. C and Z change 0 JMP addr (jump) Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the same E JCC addr (jump if carry set) Jumps to the instruction stored in address addr if C=0. If C is set, (jump if carry not set) 10 JEQ addr Jumps to the instruction stored in address addr if Z=1. If Z is not	3	ADDA addr	Adds the value in memory location addr to the value in ACCA and
(subtract value in memory from ACCA) 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 addr) 8 COMA (INCA ACCA) Replace the value in ACCA. C stays the same and Z changes 9 INCA (logical shift left ACCA) Increment value in ACCA. C stays the same and Z changes 8 LSLA (logical shift right ACCA) Logical shift left of ACCA. C and Z change 9 INCA (logical shift right ACCA) Logical shift right of ACCA. C and Z change 8 LSLA (logical shift right ACCA) Logical shift right of ACCA. C and Z change 9 ISRA (logical shift right ACCA) Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the same 0 JMP 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 <t< td=""><td></td><td>(add ACCA and value in memory)</td><td>saves the result in ACCA. C and Z change</td></t<>		(add ACCA and value in memory)	saves the result in ACCA. C and Z change
ACCA) Perform a logical AND of the value in memory location addr with (logical AND of ACCA and value in memory) 6 ORAA addr (logical CA 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 memory) Perform a logical OR of the value in addr. C stays the same, Z changes used addr) 8 COMA (Compares ACCA to the value in addr from ACCA. ACCA does not change. C and Z change addr) 9 INCA (INCCA ACCA) Replace the value in ACCA. C stays the same and Z changes 9 INCA (INCA ACCA) Increment value in ACCA. C and Z change 18 COMA (logical shift left ACCA) Logical shift left of ACCA. C and Z change 9 INCA (logical shift right ACCA) Logical shift right of ACCA. C and Z change 19 ISRA (logical shift right ACCA) Logical shift right of ACCA. C and Z change 10 JMP addr (Journa) Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the same 10 JMP addr (Journa) Jumps to the instruction. C and Z stay the same 11 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 10 JEQ addr Jumps to the instruction stored in a	4	SUBA addr	Subtracts the value in memory location addr from the value in
ACCA) Perform a logical AND of the value in memory location addr with (logical AND of ACCA and value in memory) 6 ORAA addr (logical CA 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 memory) Perform a logical OR of the value in addr. C stays the same, Z changes used addr) 8 COMA (Compares ACCA to the value in addr from ACCA. ACCA does not change. C and Z change addr) 9 INCA (INCCA ACCA) Replace the value in ACCA. C stays the same and Z changes 9 INCA (INCA ACCA) Increment value in ACCA. C and Z change 18 COMA (logical shift left ACCA) Logical shift left of ACCA. C and Z change 9 INCA (logical shift right ACCA) Logical shift right of ACCA. C and Z change 19 ISRA (logical shift right ACCA) Logical shift right of ACCA. C and Z change 10 JMP addr (Journa) Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the same 10 JMP addr (Journa) Jumps to the instruction. C and Z stay the same 11 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 10 JEQ addr Jumps to the instruction stored in a		(subtract value in memory from	
Image: Construction of ACCA and value in memory)the value in ACCA. Save the result in ACCA. C stays the same, Z changes6ORAA 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 changes7CMPA addr (Compares ACCA to the value in addrCompare ACCA to value in addr. This is done by subtracting the value in addr from ACCA. ACCA does not change. C and Z change8COMA (Complement ACCA)Replace the value in ACCA. C stays the same and Z changes9INCA (INCA ACCA)Increment value in ACCA. C stays the same and Z changes9INCA (logical shift left ACCA)Logical shift left of ACCA. C and Z changeCASRA (logical shift right ACCA)Logical shift right of ACCA. C and Z changeCASRA (Arithmetic shift right ACCA)Arithmetic shift right of ACCA. C and Z changeDJMP addr (jump)Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the sameEJCC addr (jump if carry 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 same10JEQ addrJumps to the instruction stored in address addr if Z=1. If Z is not		ACCA)	
Image: Construction of ACCA and value in memory)the value in ACCA. Save the result in ACCA. C stays the same, Z changes6ORAA 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 changes7CMPA addr (Compares ACCA to the value in addrCompare ACCA to value in addr. This is done by subtracting the value in addr from ACCA. ACCA does not change. C and Z change8COMA (Complement ACCA)Replace the value in ACCA. C stays the same and Z changes9INCA (INCA ACCA)Increment value in ACCA. C stays the same and Z changes9INCA (logical shift left ACCA)Logical shift left of ACCA. C and Z changeCASRA (logical shift right ACCA)Logical shift right of ACCA. C and Z changeCASRA (Arithmetic shift right ACCA)Arithmetic shift right of ACCA. C and Z changeDJMP addr (jump)Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the sameEJCC addr (jump if carry 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 same10JEQ addrJumps to the instruction stored in address addr if Z=1. If Z is not	5	ANDA addr	Perform a logical AND of the value in memory location addr with
memory)changes6ORAA 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 changes7CMPA 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 change8COMA (Complement ACCA)Replace the value in ACCA with its one's complement. C is set to 1 and Z changes9INCA (INCA ACCA)Increment value in ACCA. C stays the same and Z changes9INCA (logical shift left ACCA)Logical shift left of ACCA. C and Z changeBLSRA (logical shift right ACCA)Logical shift right of ACCA. C and Z changeCASRA (Arithmetic shift right ACCA)Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the sameEJCS addr (jump)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 same10JEQ addrJumps to the instruction stored in address addr if C=0. If C is set, continue with next instruction. C and Z stay the same		(logical AND of ACCA and value in	
Image: Constraint of ACCA and value in memoryvalue in ACCA. Save the result in ACCA. C stays the same, Z changes7CMPA 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 change8COMA (Complement ACCA)Replace the value in ACCA. ACCA does not change. C and Z changes9INCA (INCA ACCA)Increment value in ACCA. C stays the same and Z changes9INCA (INCA ACCA)Logical shift left of ACCA. C and Z changeBLSRA (logical shift right ACCA)Logical shift right of ACCA. C and Z changeCASRA (Arithmetic shift right ACCA)Arithmetic shift right of ACCA. C and Z changeDJMP addr (jump)Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the sameEJCS addr (jump if carry set)Jumps to the instruction stored in address addr if C=0. If C is not set, continue with next instruction. C and Z stay the same10JEQ addrJumps to the instruction stored in address addr if Z=1. If Z is not		memory)	changes
Image: Constraint of ACCA and value in memoryvalue in ACCA. Save the result in ACCA. C stays the same, Z changes7CMPA 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 change8COMA (Complement ACCA)Replace the value in ACCA. ACCA does not change. C and Z changes9INCA (INCA ACCA)Increment value in ACCA. C stays the same and Z changes9INCA (INCA ACCA)Logical shift left of ACCA. C and Z changeBLSRA (logical shift right ACCA)Logical shift right of ACCA. C and Z changeCASRA (Arithmetic shift right ACCA)Arithmetic shift right of ACCA. C and Z changeDJMP addr (jump)Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the sameEJCS addr (jump if carry set)Jumps to the instruction stored in address addr if C=0. If C is not set, continue with next instruction. C and Z stay the same10JEQ addrJumps to the instruction stored in address addr if Z=1. If Z is not	6	ORAA addr	Perform a logical OR of the value in memory location addr with the
memory)7CMPA 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 change8COMA (Complement ACCA)Replace the value in ACCA. ACCA does not change. C and Z changes9INCA (INCA ACCA)Increment value in ACCA. C stays the same and Z changes9INCA (logical shift left ACCA)Logical shift left of ACCA. C and Z changeBLSRA (logical shift right ACCA)Logical shift right of ACCA. C and Z changeCASRA (Arithmetic shift right ACCA)Arithmetic shift right of ACCA. C and Z changeDJMP addr (jump)Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the sameEJCS addr (jump if carry set)Jumps to the instruction. C and Z stay the sameFJCC addr (jump if carry not set)Jumps to the instruction. C and Z stay the same10JEQ addrJumps to the instruction stored in address addr if C=0. If C is not set, continue with next instruction. C and Z stay the same		(logical OR of ACCA and value in	S S S S S S S S S S S S S S S S S S S
(Compares ACCA to the value in addr)value in addr from ACCA. ACCA does not change. C and Z change8COMA (Complement ACCA)Replace the value in ACCA with its one's complement. C is set to 1 and Z changes9INCA (INCA ACCA)Increment value in ACCA. C stays the same and Z changesALSLA (logical shift left ACCA)Logical shift left of ACCA. C and Z changeBLSRA (logical shift right ACCA)Logical shift right of ACCA. C and Z changeCASRA (Arithmetic shift right ACCA)Arithmetic shift right of ACCA. C and Z changeDJMP addr (jump)Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the sameEJCS 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 same10JEQ addrJumps to the instruction stored in address addr if Z=1. If Z is not			
(Compares ACCA to the value in addr)value in addr from ACCA. ACCA does not change. C and Z change8COMA (Complement ACCA)Replace the value in ACCA with its one's complement. C is set to 1 and Z changes9INCA (INCA ACCA)Increment value in ACCA. C stays the same and Z changesALSLA (logical shift left ACCA)Logical shift left of ACCA. C and Z changeBLSRA (logical shift right ACCA)Logical shift right of ACCA. C and Z changeCASRA (Arithmetic shift right ACCA)Arithmetic shift right of ACCA. C and Z changeDJMP addr (jump)Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the sameEJCS 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 same10JEQ addrJumps to the instruction stored in address addr if Z=1. If Z is not	7	CMPA addr	Compare ACCA to value in addr. This is done by subtracting the
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 same 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 Jumps to the instruction stored in address addr if Z=1. If Z is not		(Compares ACCA to the value in	
(Complement ACCA)and Z changes9INCA (INCA ACCA)Increment value in ACCA. C stays the same and Z changesALSLA (logical shift left ACCA)Logical shift left of ACCA. C and Z changeBLSRA (logical shift right ACCA)Logical shift right of ACCA. C and Z changeCASRA (Arithmetic shift right ACCA)Arithmetic shift right of ACCA. C and Z changeDJMP addr (jump)Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the sameEJCS 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 sameFJCC 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 same10JEQ addrJumps to the instruction stored in address addr if Z=1. If Z is not		addr)	
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 same 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 Jumps to the instruction stored in address addr if Z=1. If Z is not	8	COMA	Replace the value in ACCA with its one's complement. C is set to 1
(INCA ACCA)Increment value in ACCA. C stays the same and Z changesALSLA (logical shift left ACCA)Logical shift left of ACCA. C and Z changeBLSRA (logical shift right ACCA)Logical shift right of ACCA. C and Z changeCASRA (Arithmetic shift right ACCA)Arithmetic shift right of ACCA. C and Z changeDJMP addr (jump)Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the sameEJCS 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 sameI0JEQ addrJumps to the instruction stored in address addr if Z=1. If Z is not		(Complement ACCA)	
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 same E JCS addr (jump if carry set) Jumps to the instruction stored in address addr if C=0. 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 Jumps to the instruction stored in address addr if Z=1. If Z is not	9	INCA	Incompart value in ACCA C stars the same and 7 shares
Image: Construction bigLogical shift left of ACCA. C and Z changeBLSRA (logical shift right ACCA)Logical shift right of ACCA. C and Z changeCASRA (Arithmetic shift right ACCA)Arithmetic shift right of ACCA. C and Z changeDJMP addr (jump)Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the sameEJCS 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 sameFJCC 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 same10JEQ addrJumps to the instruction stored in address addr if Z=1. If Z is not		(INCA ACCA)	Increment value in ACCA. C stays the same and Z changes
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 same 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 Jumps to the instruction stored in address addr if Z=1. If Z is not	Α	LSLA	Logical shift left of ACCA. C and Z shapped
CASRA (Arithmetic shift right ACCA)Arithmetic shift right of ACCA. C and Z changeDJMP addr (jump)Jumps to the instruction stored in address addr. The PC is replaced with addr. C and Z stay the sameEJCS 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 sameFJCC 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 same10JEQ addrJumps to the instruction stored in address addr if Z=1. If Z is not		(logical shift left ACCA)	Logical shift left 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 same 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 Jumps to the instruction stored in address addr if Z=1. If Z is not	В	LSRA	Logical shift right of ACCA C and Z shapped
(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 same 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 Jumps to the instruction stored in address addr if Z=1. If Z is not		(logical shift right ACCA)	Logical shift fight of ACCA. C and Z Change
Image: Construction of the construc	C	ASRA	Anithmatic shift night of ACCA C and Z shares
(jump) replaced with addr. C and Z stay the same 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 Jumps to the instruction stored in address addr if Z=1. If Z is not		(Arithmetic shift right ACCA)	Arithmetic shift right of ACCA. C and Z change
E JCS addr 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 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 Jumps to the instruction stored in address addr if Z=1. If Z is not	D	JMP addr	Jumps to the instruction stored in address addr. The PC is
(jump if carry set) 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 Jumps to the instruction stored in address addr if Z=1. If Z is not		(jump)	replaced with addr. C and Z stay the same
(jump if carry set) 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 Jumps to the instruction stored in address addr if Z=1. If Z is not	E	JCS addr	Jumps to the instruction stored in address addr if C=1. If C is not
(jump if carry not set) continue with next instruction. C and Z stay the same 10 JEQ addr Jumps to the instruction stored in address addr if Z=1. If Z is not			-
(jump if carry not set) continue with next instruction. C and Z stay the same 10 JEQ addr Jumps to the instruction stored in address addr if Z=1. If Z is not	F	JCC addr	Jumps to the instruction stored in address addr if $C=0$. If C is set,
		(jump if carry not set)	
	10	JEQ addr	Jumps to the instruction stored in address addr if Z=1. If Z is not
		(jump if Z set)	-

The outputs of the control unit 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 implement 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, 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 like this the address to the memory will be from the PC, i.e., 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 be incremented to the next memory address. What the control unit does next will depend on the data loaded into the INST register. Here is a sample code of how you may structure your module.

Example 1

Consider the instruction LDAA addr where addr= $0 \times F5$. We will further 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	
\rightarrow	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, 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 Add	ress Memory Data
	80	01
\rightarrow	81	F5
	82	Next instruction
INST	= 01 (LDAA ad	ldr 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 $0 \times F5$ is placed on the DATA line. Then $0 \times F5$ 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

EX2: Now that MAR contains the value $0 \times F5$, the multiplexer should select MAR as the source of the address. This address should then be read which causes the memory contents of address $0 \times F5$ 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

PC	Memory A	Address	Memory Data	
	80		01	
	81		F5	
\rightarrow	02		Next instruction	
INST	= 01 (LDA	A addr	op code)	
MAR	= F5			

 $0 \times F5$ will be on the address lines of the memory unit, and the data lines out of the memory will contain the data in address $0 \times F5$. 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 $\#num=0 \times F5$. 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
\rightarrow	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
\rightarrow		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 $0 \times F5$ 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 $0 \times F5$. 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 $addr=0 \times F5$. Assume the op code for JMP addr is 0×12 . Before the program begins, the situation is as below:

PC	Memory Address	Memory Data
\rightarrow	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	Men	nory A	Addres	ss Men	nory Data
		84			12
\rightarrow		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 $0 \times F5$ 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 $0 \times F5$). 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.

1 Prelab

- 1. The output of the control unit depends on both the present state and the input. What type of state machine is this.
- 2. Draw the state diagram for the control unit.

2 Lab

- 1. Assign op codes to each instruction in the instruction set.
- 2. Write a Verilog program to implement the control unit.
 - To improve readability you should 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.
- 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 both when the carry is set and when the carry is not set.