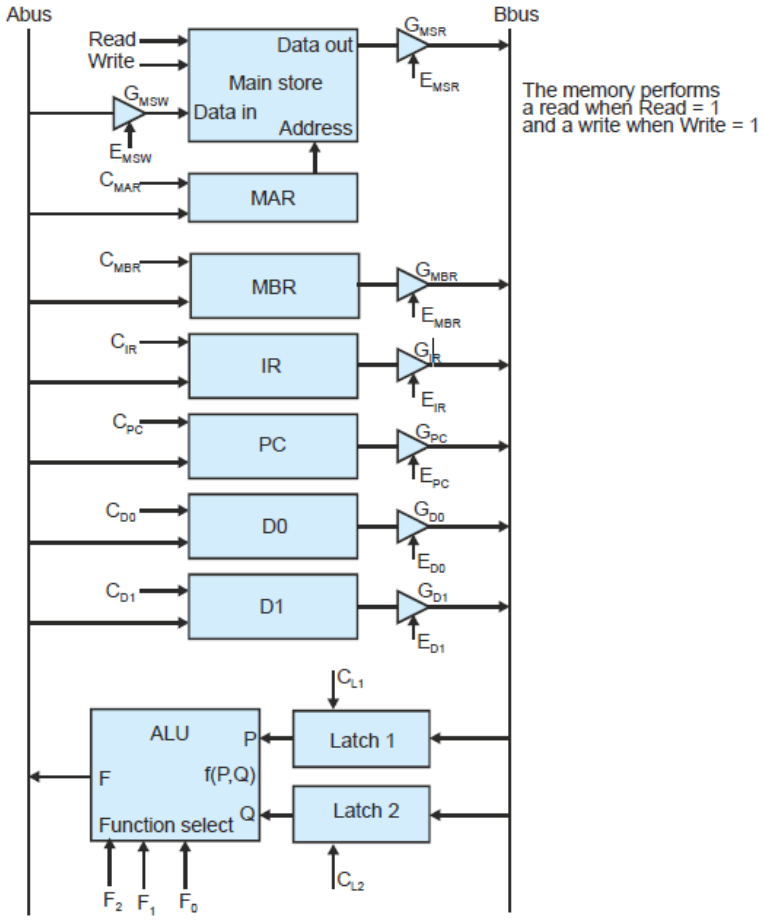
The University of Alabama in Huntsville
ECE Department
CPE 221 01
Fall 2019
Homework #5 Solution

7.1 For the microprogrammed architecture of Figure P7.1, give the sequence of actions required to implement the instruction ADD D0, D1 which is defined in RTL as D1 ← D1 + D0. You should describe the actions that occur in plain English (e.g., "Put data from this register on that bus") and as a sequence of events (e.g., Read = 1, E_{MSR}). The following table defines the effect of the ALU's function code. Note that all data has to pass through the ALU (the copy function) to get from bus B to bus A.



To perform the addition D1 must be latched into an ALU latch, D2 latched into an ALU latch, the ALU set to add and the result latched into D1. That is,

Cycle 1: $E_{D0} = 1, C_{L1}$ **;D0 or D1 in any order and use latch L1 or latch L2**

Cycle 2: $E_{D1} = 1, C_{L2}$ **;copy D1 into latch 2**

Cycle 3: $ALU(f_2,f_1,f_0) = 1,1,0, C_{D1}$ **;perform addition and latch result in D1.**

F_2	F_1	F_0	Operation	
0	0	0	Copy P to bus A	A = P
0	0	1	Copy Q to bus A	A = Q
0	1	0	Copy P + 1 to bus A	A = P + 1
0	1	1	Copy Q + 1 to bus A	A = Q + 1
1	0	0	Copy P - 1 to bus A	A = P – 1
1	0	1	Copy Q - 1 to bus A	A = Q – 1
1	1	0	Copy bus P + Q to bus A	A = P + Q
1	1	1	Copy bus P - Q to bus A	A = P – Q

7.2 For the architecture of Figure P7.1 write the sequence of signals and control actions necessary to implement the fetch cycle.

The fetch cycle involves reading the data at the address in the PC, moving the instruction read from memory to the IR, and updating the PC.

Cycle 1: $E_{PC} = 1, C_{L1}$ **;move PC to latch 1**

Cycle 2: $ALU(f_2,f_1,f_0) = 0,0,0, C_{MAR}$ **;pass PC through ALU and clock into MAR ;the PC is in L1 so we can increment it**

Cycle 3: $ALU(f_2,f_1,f_0) = 0,1,0, C_{PC}$ **;use the ALU to increment L1 and move to PC**

Cycle 4: $Read = 1, E_{MSR} = 1, C_{L1}$ **;move instruction from memory to latch 1**

Cycle 5: $ALU(f_2,f_1,f_0) = 0,0,0, C_{IR}$ **;pass instruction through ALU and clock into IR**

7.5 For the architecture of Figure P7.1, write the sequence of signals and control actions necessary to execute the instruction ADD M, D0 that adds the contents of memory location M to data register D0 and deposits the result in D0. Assume that the address M is in the instruction register IR.

This instruction requires a memory read followed by an addition.

Cycle 1: $E_{IR} = 1, C_{L1}$ **;move IR (i.e., address) to latch 1**

Cycle 2: $ALU(f_2,f_1,f_0) = 0,0,0, C_{MA}R$ **;pass IR through ALU and clock into MAR**

Cycle 3: $Read = 1, E_{MSR} = 1, C_{L1}$ **;move data from memory to latch 1**

Cycle 4: $E_{D0} = 1, C_{L2}$ **;move D0 to latch 2**

Cycle 5: $ALU(f_2,f_1,f_0) = 1,1,0, C_{D0}$ **;perform addition and clock result into D0**

7.6 This question asks you to implement *register indirect addressing*. For the architecture of Figure P7.1, write the sequence of signals and control actions necessary to execute the instruction ADS (D1), D0 that adds the contents of the memory location pointed to by the contents of register D1 to register D0 and deposits the result in D0. This instruction is defined in RTL form as D0 ← M[D1] + D0

Here, we have to read the contents of a register, use it as an address, and read from memory.

Cycle 1: $E_{D1} = 1, C_{L1}$ **;move D1 (i.e., address) to latch 1**

Cycle 2: $ALU(f_2,f_1,f_0) = 0,0,0, C_{MAR}$ **;pass D1 (the pointer) through ALU and into MAR**

Cycle 3: $Read = 1, E_{MSR} = 1, C_{L1}$ **;move data from memory to latch 1 (actual data)**

Cycle 4: $E_{D0} = 1, C_{L2}$ **;move D0 to latch 2**

Cycle 5: $ALU(f_2,f_1,f_0) = 1,1,0, C_{D0}$ **;perform addition and clock result into D0**

7.7 This question asks you to implement *memory indirect addressing*. For the architecture of Figure P7.1, write the sequence of signals and control actions necessary to execute the instruction ADD [M], D0 that adds the contents of the memory location pointed to by the contents of memory location M to register D0 and deposits the result in D0. This instruction is defined in RTL form as D0 ← M[M] + D0

We have to read the contents of a memory location, use it as an address, and read from memory. We can begin with the same code we used for ADD M,D0.

Cycle 1: E_{IR} = 1, C_{L1} ;move IR (i.e., address) to latch 1
Cycle 2: ALU(f_2,f_1,f_0) = 0,0,0, C_{MAR} ;pass IR through ALU and clock into MAR
Cycle 3: Read = 1, E_{MSR} = 1, CL1 ;move data from memory to latch 1 (a pointer)
Cycle 4: ALU(f_2,f_1,f_0) = 0,0,0, C_{MAR} ;pass the pointer through ALU and clock into MAR
Cycle 5: Read = 1, E_{MSR} = 1, C_{L1} ;move data from memory to latch 1 (the data)
Cycle 6: E_{D0} = 1, C_{L2} ;move D0 to latch 2
Cycle 7: ALU(f_2,f_1,f_0) = 1,1,0, C_{D0} ;perform addition and clock result into D0

7.8 This question asks you to implement *memory indirect addressing with index*. For the architecture of Figure P7.1, write the sequence of signals and control actions necessary to execute the instruction ADD [M, D1], D0 that adds the contents of the memory location pointed at by the contents of the memory location M plus the contents of register D1 to register D0 and deposits the result in D0. This instruction is defined in RTL form as D0 ← M[M + D1] + D0

We have to read the contents of a memory location, generate an address by adding this to a data register, and then use the sum to get the actual data. We can begin with the same code we used for ADD [M],D0.

Cycle 1: E_{IR} = 1, C_{L1} ;move IR (i.e., address) to latch 1
Cycle 2: ALU(f_2,f_1,f_0) = 0,0,0, C_{MAR} ;pass IR through ALU and clock into MAR
Cycle 3: Read = 1, E_{MSR} = 1, C_{L1} ;move data from memory to latch 1 (this is a pointer)
Cycle 4: E_{D1} = 1, C_{L2} ;move D1 to latch 2
Cycle 5: ALU(f_2,f_1,f_0) = 1,1,0, C_{MAR} ;add to get the indexed address, clock into MAR
Cycle 6: Read = 1, E_{MSR} = 1, C_{L1} ;move data from memory to latch 1 (this is the data)
Cycle 7: E_{D0} = 1, C_{L2} ;move D0 to latch
Cycle 8: ALU(f_2,f_1,f_0) = 1,1,0, C_{D0} ;perform addition and clock result into D0

Extra Write the code to implement the expression A = ((B * (C – D))/E + F) * G on 3-, 2-, 1-, and 0-address machines. Do not rearrange the expression. In accordance with programming language practice, computing the expression should not change the values of its operands. When working with 0-address instructions, assume that the operation is TOS ← SOS OP TOS.

3-address	2-address	1-address	0-address
SUB A, C, D	LOAD A, C	LDA C	PUSH C
MPY A, A, B	SUB A, D	SUB D	PUSH D
DIV A, A, E	MPY A, B	MPY B	SUB
ADD A, A, F	DIV A, E	DIV E	PUSH B
MPY A, A, G	ADD A, F	ADD F	MPY
	DIV A, G	MPY G	PUSH E
		STA A	DIV
			PUSH F
			ADD
			PUSH G
			MPY
			POP A