Homework 7 solution

1. Give a complete description of the instruction:
   `x31A1:  LDI R1, label` `; where label corresponds to the address x3246`

   Assemble LDI R1, label:

   First, PCoffset9 = least significant 9 bits of the "distance" between the PC and the label:
   label - (PC) => x3246 - (x31A1 + x0001) = x3246 - x31A2 = x00A4 => **PCoffset9 = 0 1010 0100**
   SO:
   `1010 001 0 1010 0100 => xA2A4`

   RT description of LDI DR, label:

   DR <- Mem[Mem[(PC) + SEXT(IR[8:0])]] *where SEXT(IR[8:0]) => x00A4*

   OR:

   EA = Mem[(PC) + SEXT(PCoffset9)]

   DR <- Mem[EA]

   Meaning: The destination register receives the value stored at the memory location whose address is the value stored at "label" *(i.e. the Effective Address is the value stored at label).*

   The sequence of control signals: *see schematic - note that some of the circuit elements mentioned below are not shown on the simplified versions (p. 81 & 142 of the text, and in Piazza) - e.g. in steps 6 & 8*

   a) MAR <- EA: Calculate EA and write it to MAR *(same as for STI):*

   1.a ADDR1MUX selects PC

   1.b ADDR2MUX selects SEXT(IR[8:0]) - i.e. PCoffset9

   (Address adder adds these two values - no control signal involved)

   2. MARMUX selects o/p of address adder
   3. Assert GateMARMUX *(write the EA to the bus)*
   4. Assert LD.MAR *(address of label now stored in MAR)*
   5. MEM.EN/R *(Read contents of label)*

      Wait for R (ready signal) from Memory to FSM before next step

   6. MDRMUX selects Memory *(this component is not shown on the simplified schematic)*
   7. Assert LD.MDR *(capture **EA** in MDR, for transfer to MAR)*
   8. Assert GateMDR
   9. Assert LD.MAR *(We now have the EA in the MAR)*

   b) DR <- Mem[(MAR)]: Read data from memory, copy to DR *(same as for LD, LDR):*

   10. DRMUX selects IR[11:9] *(this is the default, and can be omitted)*
   11. MEM.EN/R *(read data at EA - second time around the memory loop!)*

       Wait for R (ready signal) from Memory to FSM before next step

   12. MDRMUX selects Memory
   13. Assert LD.MDR *(capture **data** in MDR this time!)*
   14. Assert Gate.MDR *(write MDR value to bus)*
   15. Assert LD.REG *(write the value on the bus to the register selected by the DR decoder)*

2. Give a complete description of the instruction:

   `STR R0, R6, x0`

   Assemble `STR R0, R6, x0`:

   `0111 000 110 00 0000 => x7180`

   RT description of `STR R0, R6, x0`:

   `Mem[ (R6) + SEXT(IR[5:0]) ] <- (R0)`

   i.e. the contents of the source register (R0) are written to the memory location (EA) calculated by adding the sign-extended 6-bit 2's complement offset (x0000, in this case) to the contents of the Base Register (R6)

   The sequence of control signals:

   a) MAR <- EA: Calculate EA and write it to MAR *(same as for LDR):*

   0. SR1MUX selects BaseReg IR[8:6], R6 in this case *(this is the default, and <u>can be omitted</u>)*
   1.a ADDR1MUX selects <u>SourceRegister1 bus</u> *(which will carry the contents of the register selected by IR[8:6], i.e. the BaseReg R6)*
   1.b ADDR2MUX selects SEXT(IR[5:0]) - i.e. offset6, x0000 in this case
   (Address adder adds these two values - no control signal involved)
   2. MARMUX selects o/p of address adder
   3. Assert GateMARMUX *(write the EA to the bus)*
   4. Assert LD.MAR *(address from BaseReg R6 now stored in MAR)*

   b) MDR <- (SR): Write to MDR the contents of the Source register (R0), which for Store instructions is specified by IR[11:9]

   5. SR1MUX selects <u>IR[11:9]</u> *(this is **<u>NOT</u>** the default, and <u>MUST be shown</u>!)*
   6. ALUK selects "passs through"
   7. Assert Gate.ALU
   8. MDRMUX selects bus
   9. Assert Ld.MDR

   c) Mem[(MAR)] <- (MDR):

   10. MEM.EN/W

3. Give a complete description of the instruction: *(note that this label comes <u>before</u> the instruction!)*

   `x35D0: BRzp label` `; where label corresponds to address x34EF`

   Assemble BRzp label:

   As in Q. 1, PCoffset9 = label - (PC) = x34EF - (x35D0 + x0001) = x34EF - x35D0, which is <u>negative</u>!
   i.e. -(PCoffset9) = -(x34EF - x35D0) = x35D0 - x34EF = x00E2 => 0 1110 0010
   Thus PCoffset9 = -(0 1110 0010) i.e. the 2's complement = **1 0001 1110**
   SO:

   `0000 011 1 0001 1110 => x071E`

   RT description of BRzp label:

   PC <- (PC) + SEXT(IR[8:0]) **<u>iff Z or P</u>** *(i.e. if either the Z or the P condition codes are currently set)*

The sequence of control signals:

1.a ADDR1MUX selects PC

1.b ADDR2MUX selects SEXT(IR[8:0]) - i.e. PCoffset9

2. PCMUX selects o/p of address adder

3. Assert Ld.PC iff Z or P

The actual Condition Code test is: **n.N + z.Z + p.P**

where **n, z, p** are IR[11:9], the bits that specify which of the Condition Codes **N, Z, P** are to be tested.

==========

5.15

```
1110 001 000100000  ( LEA R1, 0x20 )    R1 <- x3121
0010 010 000100000  ( LD R2, 0x20 )     R2 <- Mem[0x3122] = x4566
1010 011 000100001  ( LDI R3, 0x20 )    R3 <- Mem[Mem[0x3123]] = xABCD
0110 100 010 000001 ( LDR R4, R2, x1 ) R4 <- Mem[R2 + 0x1] = xABCD
1111 0000 0010 0101 ( TRAp 0x25 )       = HALT
```

5.19

((PC) – 64) to ((PC) + 63)
(PC) is the incremented PC value.

5.20

We would now require 6 bits for the PCoffset.

5.24

The largest address this instruction can load from is x4011 + x1F = x4030, as in the instruction:
LDR R5, R4, x1F
If the LDR offset were zero-extended, the largest address the LDR instruction can load from is x4011 + x3F = x4050 and the smallest address is x4011 + x0 = x4011

5.29

(a) LDR R2, R1, #0 ; load R2 with contents of location pointed to by R1
STR R2, R0, #0 ; store those contents into location pointed to by R0

(b) The required "micro-operations" are:
MAR <= (SR) *(via the global bus)*
MDR <= Mem[MAR]
MAR <= (DR) *(via the global bus)*
Mem[MAR] <= MDR