Tu Lam

ECE 375 / Dr. Justin Goins

Oct 24th, 2020

Homework #2

(Due Date: Oct 28th, 2020)

2. *Consider the following section of AVR assembly code (from the previous problem) with its equivalent (partially completed) address and binaries on the right.*

a) ***KKKK dddd KKKK(@ address $0047)***

At the address $0047, the address is located at the set of instruction [LDI R20, var1] and the result are:

$KKKK\ dddd\ KKKK$ where the ($1\ dddd = 1\ 0100$) and ($KKKK\ KKKK = 0000\ 1110$)

b) ***rd dddd rrrr (@ address $004B)***

At the address $004B, the address is located at the set of instruction [CP R20, R21] and the result are:

$rd\ dddd\ rrrr$ where the ($d\ dddd = 1\ 0100$) and ($r\ rrrr = 1\ 0101$)

c) ***kk kkkk k (@ address $004C)***

At the address $004C, the address is located at the set of instruction [BRLT finish] and the result are:

$kk\ kkkk\ k$ where the ($kk\ kkkk\ k = 00\ 0010\ 0$)

The ‘finish’ is at the address $0050 and BRLT is at the address $004C and we have ($0050 - $004C = $0004)

d) ***d dddd (@ address $004E)***

At the address $004E, the address is located at the set of instruction [INC count] and the result are:

$d\ dddd$ where the ($d\ dddd = 0\ 0111$)

The ‘count’ label is holding R7

e) ***r rrrr (@ address $0051)***

At the address $0051, the address is located at the set of instruction [ST Y+, R20] and the result are:

$r\ rrrr$ where the ($r\ rrrr = 1\ 0100$)

3. *The term single-port means that only one register value can be read or written at a time. Suppose the pseudo-CPU is used to implement the AVR instruction ST -Y, R14. Give the sequence of microoperations that would be required to Fetch and Execute this instruction. Note that this instruction occupies 16-bits. Your solution should result in exactly 6 cycles for the fetch cycle and no more than 7 cycles for the execute cycle. You may assume only the AC and PC registers have the capability to increment/decrement themselves. Assume the MDR register is only 8 bits wide (which implies that memory is organized into consecutive addressable bytes), and AC, PC, IR, and MAR are 16-bits wide. Also, assume the Internal Data Bus is 16 bits wide and thus can handle 8-bit or 16-bit (as well as portions of 8-bit or 16-bit) transfers in one microoperation. Clearly state any other assumptions made.*

Fetch Cycle:

Step 1: $MAR \leftarrow PC$	//Move the (HIGH) into MAR
Step 2: $MDR \leftarrow M(MAR),\ PC \leftarrow PC + 1$	//Increment PC & read instruction
Step 3: $IR(15 \ldots 8) \leftarrow MDR$	//Move data into instruction register
Step 4: $MAR \leftarrow PC$	//Move the (LOW) into MAR
Step 5: $MDR \leftarrow M(MAR),\ PC \leftarrow PC + 1$	//Increment PC & read instruction
Step 6: $IR(7 \ldots 0) \leftarrow MDR$	//Move data into instruction register

Execute Cycle:

Step 1: $MAR(15 \ldots 8) \leftarrow R14$	//Move R14(HIGH) register address into MAR
Step 2: $MAR(7 \ldots 0) \leftarrow R14$	//Move R14(LOW) register address into MAR
Step 3: $MDR \leftarrow M(MAR)$	//Move the data from M(Y) to MDR
Step 4: $AC \leftarrow MAR$	//Move the address from MAR into AC
Step 5: $AC \leftarrow AC - 1$	//Decrement the Y register
Step 6: $R29 \leftarrow AC(15 \ldots 8)$	//Store Y(HIGH) back from AC to R29
Step 7: $R28 \leftarrow AC(7 \ldots 0)$	//Store the Y(LOW) back into R28

4. *Now imagine that we take the pseudo-CPU (from the previous problem) and add a Stack Pointer (SP). Suppose the pseudo-CPU can be used to implement the AVR instruction ICALL(Indirect Call to Subroutine) with the format shown below*

Fetch Cycle:

```
Step 1: $MAR \leftarrow PC$                              //Move the (HIGH) into MAR
Step 2: $MDR \leftarrow M(MAR),\ PC \leftarrow PC + 1$   //Increment PC & read instruction
Step 3: $IR(15 \dots 8) \leftarrow MDR$                  //Move data into instruction register
Step 4: $MAR \leftarrow PC$                              //Move the (LOW) into MAR
Step 5: $MDR \leftarrow M(MAR),\ PC \leftarrow PC + 1$   //Increment PC & read instruction
Step 6: $IR(7 \dots 0) \leftarrow MDR$                   //Move data into instruction register
```

Execute Cycle:

```
Step 1: $PC(15 \dots 8) \leftarrow R31$                  //Put Z(HIGH) into PC
Step 2: $PC(7 \dots 0) \leftarrow R30$                   //Put Z(LOW) into PC
Step 3: $MDR \leftarrow PC(7 \dots 0)$                   //Move LOW  byte into MDR
Step 4: $MAR \leftarrow SP$                              //Move stack pointer to the MAR
Step 5: $M(MAR) \leftarrow MDR,\ SP \leftarrow SP - 1$   //Push LOW byte onto the stack & decrement stack
Step 6: $MDR \leftarrow PC(15 \dots 8)$                  //Move HIGH byte into MDR
Step 7: $MAR \leftarrow SP$                              //Move new SP into MAR
Step 8: $M(MAR) \leftarrow MDR,\ SP \leftarrow SP - 1$   // Push HIGH byte onto the stack & decrement stack
```