[24 pts]

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

Determine the values for:

(a) KKKK dddd KKKK (@ address $004C)

$dddd = 1\,0001$

$KKKK\ KKKK = 0001\ 0010$

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

$d\ dddd = 1\,0100$

(c) kk kkkk k (@ address $004F) (d) d dddd (@ address $0052)

$kk\ kkkk\ k = 10\ 1000\ 1$

(e) r rrrr (@ address $0055)

$r\ rrrr = 0\ 0001$

[25 pts]

3- Consider the following hypothetical 1-address assembly instruction called "Add Then Store Indirect with Pre-increment" of the form

ATS+(x) ; M[x] ← M[x]+1, M[M[x]] ← AC+M[M[x]]

The instruction will operate using indirect addressing (as illustrated in Figure 2.4a from the textbook).

Suppose we want to implement this instruction on the pseudo-CPU discussed in class augmented with a TEMP register (as shown below). The fetch cycle has been provided below. Give the sequence of *microoperations* required to implement the execute cycle for the above ATS +(x) instruction. Be sure to combine multiple register transfer operations into the same microcycle when possible. A correct execute cycle solution should consume no more than 9 microoperations.

Assume an instruction consists of 16 bits: A 4-bit opcode and a 12-bit address. All operands are 16 bits. PC and MAR each contain 12 bits. AC, MDR, and TEMP each contain 16 bits, and IR is 4 bits. Note that the original content of AC should be preserved. Assume PC is currently pointing to the ATS instruction and only PC and AC have the capability to increment/decrement themselves.

Fetch Cycle

Cycle 1: MAR ← PC;

Cycle 2: MDR ← M[MAR], PC ← PC+1 ; Read instruction & increment PC

Cycle 3: IR ← MDR_{opcode}, MAR← $MDR_{address}$

Execute Cycle:

Cycle 1: $TEMP \leftarrow AC,\ MDR \leftarrow M[MAR]$; store AC and get EA - 1

Cycle 2: $AC \leftarrow MDR$; move EA - 1 to AC

Cycle 3: $AC \leftarrow AC + 1$; increment EA - 1

Cycle 4: $MDR \leftarrow AC$; move EA to MDE

Cycle 5: $M[MAR] \leftarrow MDR,\ MAR \leftarrow MDR$; store it back in memory location x, (M[x], or EA)

; and move EA to MAR

Cycle 6: $MDR \leftarrow M[MAR],\ AC \leftarrow TEMP$; read operand, restore AC

Cycle 7: $AC \leftarrow AC + MDR$; perform add operation

Cycle 8: $MDR \leftarrow AC$; move result (AC) to MDR

Cycle 9: $M[MAR] \leftarrow MDR,\ AC \leftarrow TEMP$; store result to memory location specified by MAR and restore AC

[24 pts]

4- In order to make the pseudo-CPU more useful, suppose that we incorporate a *single-port register file* containing

32 8-bit registers (R0-R31). 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, R3. Give the sequence of microoperations that would be required to Fetch and Execute this instruction (assuming that the AVR architecture uses the little-endian convention). 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.

Hint: Remember that the Y register is actually formed from two individual 8-bit registers. They can be accessed from the Register File.

Fetch Cycle:

Cycle 1: $MAR \leftarrow PC$; move address of instruction to be executed to MAR

Cycle 2: $MDR \leftarrow M[MAR],\ PC \leftarrow PC + 1$; read instruction and increment PC

Cycle 3: $IR_{7...0} \leftarrow MDR$; move MDR to the first 8 bits of IR

Cycle 4: $MAR \leftarrow PC$

Cycle 5: $MDR \leftarrow M[MAR],\ PC \leftarrow PC + 1$

Cycle 6: $IR_{15...8} \leftarrow MDR$; move MDR to the last 8 bits of IR

Execute Cycle:

Cycle 1: $MAR_{7...0} \leftarrow R28$; move content of R28 to Y(LOW)

Cycle 2: $MAR_{15...8} \leftarrow R28$; move content of R29 to Y(HIGH)

Cycle 3: $AC \leftarrow MAR$; move MAR (Y) to AC

Cycle 4: $AC \leftarrow AC - 1$; decrement AC (Y = Y - 1)

Cycle 5: $MAR \leftarrow AC$; move AC to MAR

Cycle 6: $M[MAR] \leftarrow R3$; store R3 to memory location specified by MAR