ECE 375 Homework 2

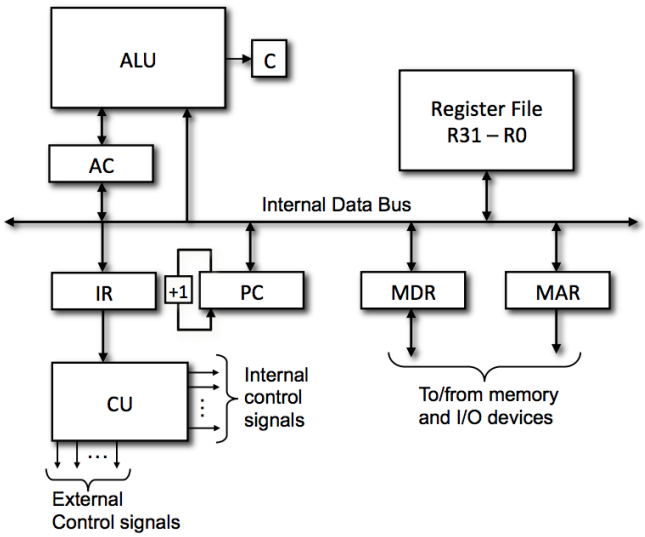
Computer Organization and Assembly Language Programming

Winter Term 2018

Jeremy Fischer

1 Homework Questions

1. Consider the internal structure of the pseudo-CPU discussed in class augmented with a *single-port register file* (i.e., only one register value can be read in a cycle) containing 32 8-bit registers (R0-R31) and a carry bit (Cbit), which is set/reset after each arithmetic operation. Suppose the pseudo-CPU can be used to implement the AVR instruction ADIW ZH:ZL,32 *(Add immediate to word)*. ADIW is a 16-bit instruction, where the upper byte represents the opcode and the lower byte represents an immediate value, i.e., "32" (do not worry about the fact that the actual format is slightly different). Give the sequence of microoperations required to Fetch and Execute the ADIW instruction. *Your solutions should result in exactly 5 cycles for the fetch cycle and 6 cycles for the execute cycle.* Assume the memory is organized into addressable bytes (i.e., each memory word is a byte), MDR, IR, and AC registers are 8-bit wide, and PC and MAR registers are 16-bit wide. Also, assume Internal Data Bus is 16-bit wide and thus can handle 8-bit or 16-bit (as well as portion of 8-bit or 16-bit) transfers in one microoperation and only PC and AC have the capability to increment itself.



Answer:

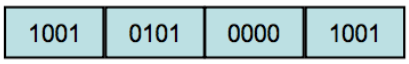
Fetch Cycle

MAR ← PC	;load the opcode address
MDR ← M[MAR]; PC ← PC + 1	;load the opcode
IR ← MDR	;load the opcode into IR
MAR ← PC	;load the immediate value address
MDR ← M[MAR]; PC ← PC + 1	;load the immediate value

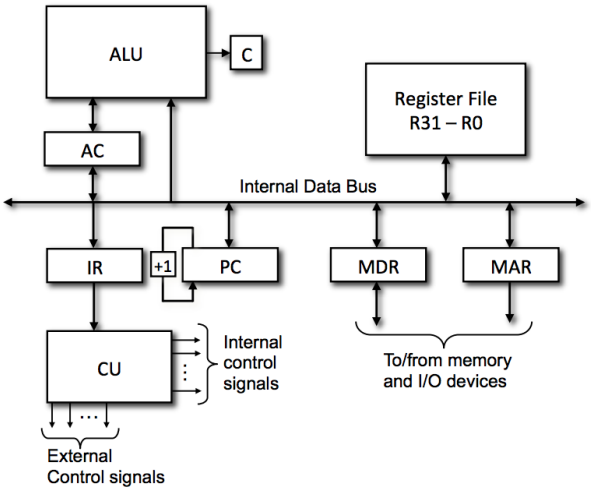
Execute Cycle

```
R16 ← AC                          ;store the old value of AC in temp register
AC ← ZL                           ;move low byte to AC
AC ← (AC + MDR), ZL ← AC          ;add low byte and immediate value, move back to ZL
AC ← ZH                           ;move high byte to AC
AC ← (AC + C), ZH ← AC            ;add high byte and carry, move back to ZH
AC ← R16                          ;restore old AC value
```

2. Consider the internal structure of the pseudo-CPU discussed in class augmented with a *single-port register file* (i.e., only one register value can be read at a time) containing 32 8-bit registers (R31-R0) and 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:



ICALL pushes the return address onto the stack and jumps to the 16-bit target address contained in the Z register. Give the sequence of microoperations required to Fetch and Execute AVR's ICALL instruction. *Your solutions should result in exactly 6 cycles for the fetch cycle and 8 cycles for the execute cycle.* Assume the memory is organized into addressable bytes (i.e., each memory word is a byte), MDR is 8 bits, and AC, SP, PC, IR, and MAR are 16 bits. Also, assume Internal Data Bus is 16-bit wide and thus can handle 8-bit or 16-bit (as well as portion of 8-bit or 16-bit) transfers in one microoperation and SP has the capability to increment/decrement itself. Clearly state any other assumptions made.



Answer:

I am assuming that the program memory is 16-bits wide, and it is possible to access the low and high byte of memory separately.

Fetch Cycle

$MAR \leftarrow PC_{low}$	;load the opcode address
$MDR \leftarrow M[MAR];$	;get the opcode low byte
$IR_{low} \leftarrow MDR$	;load the opcode low byte into IR
$MAR \leftarrow PC_{high}; PC \leftarrow PC + 1$	;load the opcode address
$MDR \leftarrow M[MAR];$	;get the opcode high byte
$IR_{high} \leftarrow MDR$	;load the opcode high byte into IR

Execute Cycle

$MAR \leftarrow SP_{low}$	;get the low byte address of stack
$MDR \leftarrow PC_{low}$	;get the low byte of return address
$M[MAR] \leftarrow MDR$	;load the low byte of ret. to low addr. of stack
$MAR \leftarrow SP_{high}$	;get the high byte address of stack
$MDR \leftarrow PC_{high}$;$SP \leftarrow SP - 1$	;get the high byte of return address and dec. stack
$M[MAR] \leftarrow MDR$	;load the high byte of ret. to high addr. of stack
$PC_{low} \leftarrow ZL$	;load low byte of jump addr. to PC low
$PC_{high} \leftarrow ZH$	;load high byte of jump addr. to PC high

3. Suppose the following array of numbers are stored in the Data Memory (represented in hexadecimal):

Address	Content
0100:	01
0101:	BE
0102:	35
0103:	EC
0104:	48
0105:	2D
0106:	04
0107:	02

(a) Assuming these numbers are signed numbers, write a subroutine using AVR assembly that (1) determines the smallest number among the 8 numbers stored in memory and (2) stores that number in the memory location $0108. Clearly comment and explain your code. Use the skeleton code shown below to implement your subroutine:

(b) Suppose these numbers are unsigned numbers (i.e., they are positive numbers). Show and explain how the code developed in part (a) would have to be modified.

Answer (a):

I am assuming that "m128def.inc" is included. Therefore, RAMEND = 0x10ff.

```
.ORG 0x0046
        ; Initialize Stack Pointer
        ldi R16, low(RAMEND)      ;Low Byte of End SRAM Address
        out SPL, R16               ;Write byte to SPL
        ldi R16, high(RAMEND)     ;High Byte of End SRAM Address
        out SPH, R16               ;Write byte to SPH

        ldi R16, 8                ;loop counter is 8
        ldi XL, low(DATA)         ;load the address of the array
        ldi XH, high(DATA)
        .DEF CURV = R17           ;current value
        .DEF CURLOW = R18         ;current lowest value
        ld CURLOW, X              ;init CURLOW to first element

        RCALL MIN ;call MIN

.ORG 0x0060
MIN:
        tst R16                   ;check if R16 is 0
        breq EXIT                 ;if it is, go to exit
        ld CURV, X+               ;get the next value in array and X++
        cp CURV, CURLOW
        brge CONT                 ;skip over changing lowest
        mov CURLOW, CURV          ;change lowest to the current value
        CONT:
                dec R16           ;else R16--
                rjmp MIN
        EXIT:
                sts $0108, CURLOW ;store lowest value in 0x0108
                ret

.DSEG
.ORG 0x0100
        DATA: .BYTE 8
        RESULT: .BYTE 1
```

Answer (b):

The only thing that would need to be changed is the *BRGE* instruction. BRGE is used for signed numbers. We would change *BRGE* for *BRSH*, since *BRSH* is for unsigned numbers. This would allow the comparisons between numbers (specifically curLow and curV) to function as we expect.

4. Determine the location (i.e., address) and binary code for each instruction in the code developed for Problem #3 part (a). Clearly explain your answers.

Answer:

Instruction	**Address**	**Binary**	**Explanation** *Black is opcode*
ldi R16, low(RAMEND)	0x0046	1110 1111 0000 1111	Rd: implied 1 = 1 R16 = 0000 K: low(0x10ff) = ff
out SPL, R16	0x0047	1011 1111 0000 1101	AA: SPL=0x3d Rr = R16 = 10000
ldi R16, high(RAMEND)	0x0048	1110 0001 0000 0000	Rd: implied 1 = 1 R16 =0000 K: high(0x10ff) = 10
out SPH, R16	0x0049	1011 1111 0000 1110	AA: SPH=0x3e Rr = R16 = 10000
ldi R16, 8	0x004A	1110 0000 0000 1000	Rd: implied 1 = 1 R16 = 0000 K: 8 = 1000
ldi XL, low(DATA)	0x004B	1110 0000 1010 0000	Rd: implied 1 = R26 = 1010 K: low(0x0100) = 00
ldi XH, high(DATA)	0x004C	1110 0000 1011 0001	Rd: implied 1 = R27 = 1011 K: low(0x0100) = 01
ld CURLOW, X	0x004D	1001 0001 0010 1100	Curlow=R18=10010
RCALL MIN	0x004E	1101 0000 0001 0001	K: $60-$4E-1=$11
tst R16	0x0060	0010 0000 0001 0000	Rd: R16 = 1000
breq EXIT	0x0061	1111 0000 0000 0110	K: $68-$60-1=$06=0110
ld CURV, X+	0x0062	1001 0001 0001 1101	Curv=R17=10001
cp CURV, CURLOW	0x0063	0001 0111 0001 0010	Rd:R17=10001 Rr:R18=10010
brge CONT	0x0064	1111 0100 0000 1100	K:$66-$64-1=$01
mov CURLOW, CURV	0x0065	0010 1111 0010 0001	Rd:R18=10010 Rr:R17=10001
dec R16	0x0066	1001 0101 0000 1010	Rd: R16 = 1 0000
rjmp MIN	0x0067	1100 1111 1111 1000	K: $60 - $67 - 1 = dec(-8)
sts $0108, CURLOW	0x0068	1001 0011 0010 0000 0000 0001 0000 1000	Rd: R18 = 10010 K: $0108
ret	0x006A	1001 0101 0000 1000	