

CS 273
Final Exam
December 5, 2005

The following exam is open book and open notes. You may feel free to use whatever additional reference material you wish, but **no electronic aids** are allowed. Please note the following instructions. There will be a ten point deduction for failure to comply with them:

- start each problem on a new sheet of paper
- write your social security number, but not your name, on each sheet of paper you turn in
- show your work whenever appropriate. There can be no partial credit unless I see how answers were arrived
- be succinct. You may lose points for facts that, while true, are not relevant to the question at hand

You have until 3:00 to finish the exam. The questions are equally weighted.

1. Arithmetic

Each of the following subparts consists of a small fragment of HC11 code, followed by a branch. For each of them, answer the following:

- Which of the A, B, X, or Y registers change as a result of executing the code? What are their new values?
- What are the values of the NZVC condition codes as a result of executing the code?
- Will the branch at the end of the code be taken?

All values except the condition codes should appear in your answers in hexadecimal.

- (a) `ldaa #27` *A gets 1b, NZVC get 000-*
`asra` *A gets 0d, NZVC gets 0011*
`bgt bogus` *N \oplus V = 1, so the branch is not taken*

I didn't realize the V bit would be set until I checked the details when I was making up the answer key myself – and it turns out the simulator gets it wrong! So I didn't take off a point for missing that one. Of course, if you don't get $V=1, Z+N\oplus V = 0$ so the branch is taken.

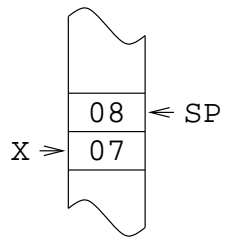
- (b) `ldaa #$d` *A gets 0d, NZVC get 000-*
`ldab #6` *B gets 06, NZVC get 000-*
`mul` *D gets 004e, NZVC get 0000*
`bra bogus` *bra is always taken*

- (c) *(the constants created with the fcb directive are not instructions. The first instruction you should interpret is the ldx)*

```
label fcb 13,1,3,-3,7
      ldx #label        X gets the address of label, NZVC get 100- (note since label must be in EEPROM, most significant bit is 1)
      ldaa #%00010001   A gets 11, NZVC get 000-
      suba 3,x          A gets 11 - (-3) = 14, NZVC get 0001
      bcs bogus         C is set, so the branch is taken
```

- (d) *(in your answer to this part, show the contents of the stack at the end, as well as the registers)*

```
lds #$ff        SP gets ff, NZVC get 000-
ldaa #7         A gets 07, NZVC get 000-
psha            07 is pushed on stack
inca            A gets 08, NZVC get 000-
psha            08 is pushed on stack
tsx             X gets sp+1
ldab 0,x        B gets 08, NZVC get 000-
blo bogus       Can't determine (oops)
```



2. Assemble the following assembly code into machine code (*the only purpose of this code is to give you several instructions to assemble. The code doesn't do anything useful*).

```

RAM      equ    0
EEPROM   equ    $f800
XYZZY    equ    7
mask     equ    $80
          org    RAM
v1       rmb    1
v2       rmb    1

          org    EEPROM
0010 f800 de 07      start ldx  XYZZY
0011 f802 97 00          staa v1
0012 f804 2e 04          bgt  bogus
0013 f806 7e f8 00      jmp  start
0014 f809 58            aslb
0015 f80a cc f8 00      bogus ldd  #start
0016 f80d 18 eb 01          addb v2,y
0017 f810 1e 07 55 ec          brset XYZZY,x %01010101 start

```

3. Consider the following HC11 machine code.

```

0007 f80a 24
0008 f80b 86 0c
0009 f80d ce f8 00
0010 f810 ab 0a
0011 f812 46
0012 f813 16
0013 f814 d7 02

```

- (a) Disassemble the code. You can leave the operands as hexadecimal “magic constants” (but you do need to have the correct addressing mode notation). The value in location \$f80a is a constant (not an instruction); it should not be decoded.

```

0007 f80a 24          fcb  $24
0008 f80b 86 0c      start ldaa #$0c
0009 f80d ce f8 00   ldx  #$f800
0010 f810 ab 0a      adda $0a,x
0011 f812 46         rora
0012 f813 16         tab
0013 f814 d7 02      stab $02

```

- (b) Suppose the code is executed. Before we begin, assume the program counter contains \$f80b, and all the other registers contain 0. After it is executed, what are the contents of the A, B, X, Y, PC, and CCR registers? You need only consider the NZVC bits of the CCR. Be sure to explain what each of the instructions does, so it'll be possible to give partial credit.

```

start ldaa #$0c      A gets $0c; NZVC get 0000
      ldx  #$f800     X gets $f800; NZVC get 1000
      adda $0a,x     A gets $30; NZVC get 0000
      rora          A gets $18; NZVC get 0000
      tab          B gets $18; NZVC get 0000
      stab $02      Location $0002 gets $18. NZVC get 0000

```

So at the end, the PC is pointing at the next instruction (\$f816), A=B=\$18, memory location \$02=\$18, X=\$f800, Y is unchanged with \$0000, and NZVC=0000

4. Compile the following into HC11 assembly code.

- (a) The following line is a fragment from the main program. You only need to compile the fragment itself; earlier code in the main program will have initialized the stack pointer. j is a global variable taking up one byte. The parameter to the function is passed on the stack.

```

j = func(7)+3;
lda #7 ; push 7

```

```

psha
jsr func ; call func
ins     ; clean up stack
adda #3 ; compute func(7) + 3
staa j  ; perform assignment

```

- (b) In compiling this code, you will need to make sure to properly set up the activation record. Proper use of indexed addressing to manipulate the parameter and local variable inside `func()` will be considered in grading this problem. ints are assumed to take one byte.

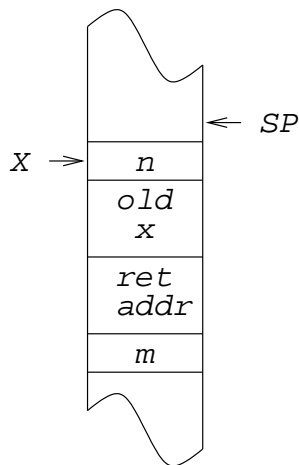
```

int func(int m)
{
    int n;
    n = 3;
    if (m > n)
        n = m + n;
    return n;
}

pshx      ; save old X
des       ; make space for n
tsx       ; point X at new activation record
ldaa #3   ; n = 3
staa 0,x  ;
cmpa 5,x  ; if (m > n)
ble ret   ;
adda 5,x  ; n = m + n
staa 0,x  ;
ret ins   ; return code -- dealloc n
pulx     ; restore x
rts      ; return n in A (it's there!)

```

Here's what the stack looks like while the function is being executed:



5. Device drivers.

(a) Write an interrupt service routine for the serial port that will do the following:

- i. Determine whether the interrupt was caused by an input character being available (RDRF) or by the transmitter being able to send a character out (TDRE) (it won't be caused by anything other than these two conditions).
- ii. If it was caused by RDRF then save the input character in a global variable `buf`, disable the receiver interrupt (RIE), and enable the transmitter interrupt (TIE).
- iii. If it was caused by TDRE, read the character from `buf`, add one to it, write the character out to the serial port, enable the receiver interrupt, and disable the transmitter interrupt.

(note: you only need to write the interrupt service routine. You may assume the serial port is properly initialized with correct settings in `BAUD`, `SCCR1`, and `SCCR2`)

```
serint  brclr   SCSR,x RDRF nordrf
        ldaa   SCDR,x
        staa  buf
        bclr  SCCR2,x RIE
        bset  SCCR2,x TIE
        bra   retint
nordrf  ldaa   buf
        inca
        staa  SCDR,x
        bset  SCCR2,x RIE
        bclr  SCCR2,x TIE
retint  rti
```

(b) From the standpoint of someone typing into the Comm window of `gdl`, what does this code do (as in, “when the user types a character...”)?

...it echoes the lexicographically next character. For most characters this will result in pretty normal behavior (type W, get X etc) but for a few (z, Z, 9...) it'll echo back something that looks unrelated but is in fact the next code in the ASCII encoding.