Comments

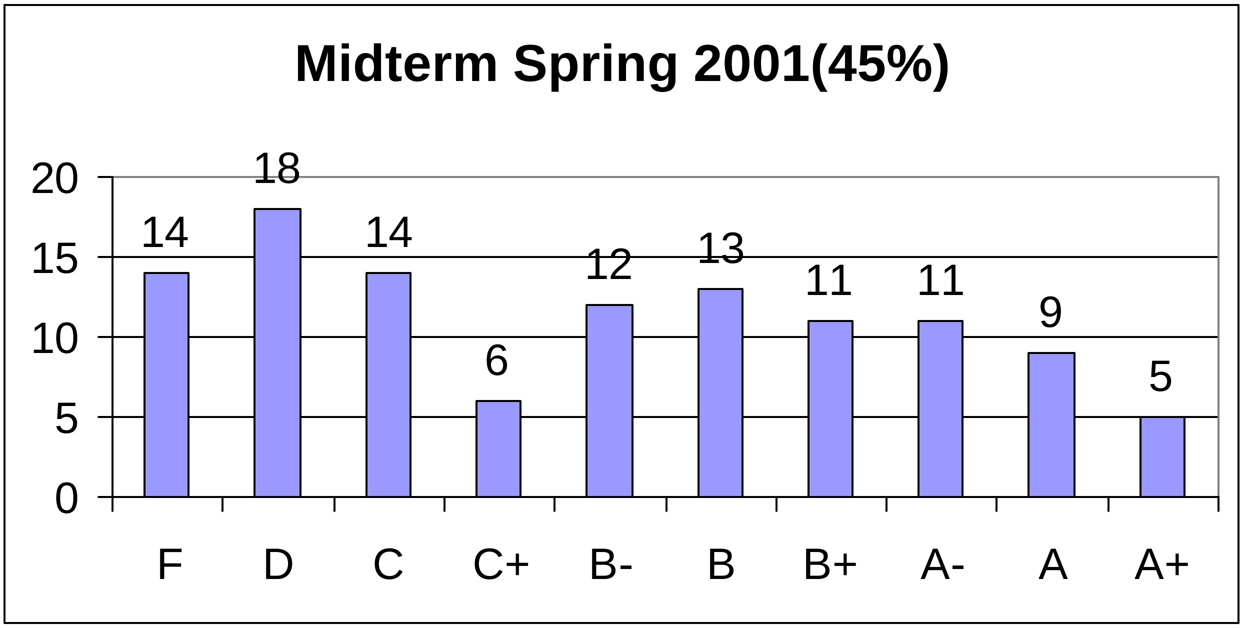
While the midterm test may have appeared to be long to some of you, the final results are quite good and in line with previous terms. Details are below.

My concern is mainly for the people who received less than 40%. I would like to encourage them to come by during my office hours and examine one-on-one what are the major items of concerns, so whatever mistakes were made they can be turned into a true learning experience and help to improve future performances. Thsi is, of course, true for everybody else as well, as I can be helpful in assess with you which mistakes may have been due to carelessness or the rush of the test, and which ones denote more a lack of understanding - both can be remedied.

The statistics are as follows: 12% failed, the average is 64% and so is the median. A few people did remarkably well, including receiving a 100% mark (congratulations, indeed! I will be glad to see you apply to become my graduate students in a few years). The distribution of marks is shown in the graph below. To see the actual table of marks, please go to the appropriate Web page where all the current marks are shown. There, I also added an extra column next to the midterm mark showing the % value - it would have taken me too long to write each one on your own papers.

Congratulations to all of you for quite a good job and I hope you will continue to study and, most of all, practice the exercises - it is invaluable to do so! Please feel free to come and see me for individual feedback.

The answers to the questions in the pages below contain my commentary (in italics) as well, to give you an idea of where I set my objectives.



C SC 230 - Computer Architecture and Assembly Language - Spring 2001

Midterm Test - February 19, 2001

INSTRUCTOR: M. Serra

STUDENT NUMBER: (last 5 digits): ______________________________

LAST NAME: _________________________SIGNATURE:____________________

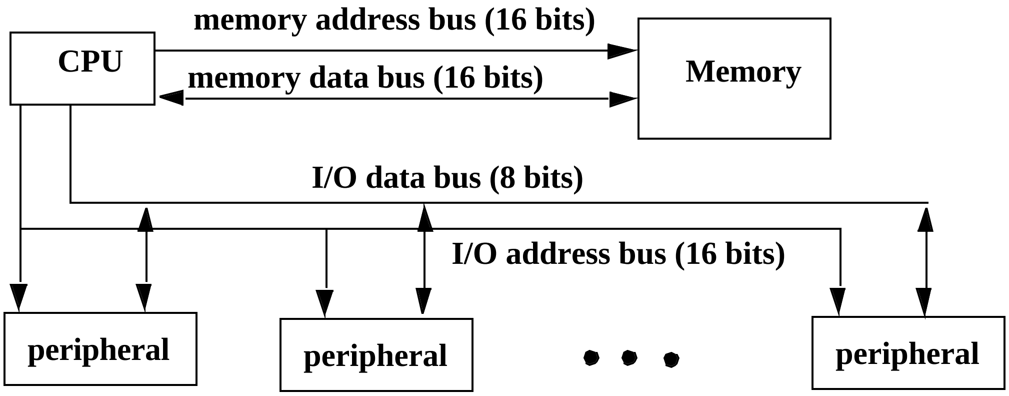
TOTAL MARKS: 98. This is a closed book test. Attempt all questions. Write your answers on the question paper in the spaces provided. Use the reverse side for rough work. Total time allowed is 60 minutes. Use ONLY the handout provided for extra notes.

QUESTION 1 [8 marks].

Thsi question was a mild permutation on 2 similar questions on assignments 1 and 2, with the variation being part (d). If you did not have those questions correct in the assignments, you really should have asked for a detailed explanation, as the concepts are fundamental.

In the picture below, an isolated I/O architecture is shown with the size of the address buses and data buses labelled. An 8-bit data bus implies here a byte-addressable cycle, while a 16-bit data bus implies here a word-addressable cycle.

(a) How many addressable locations are available for memory?	2^{16}
(b) What is the total memory size in bytes?	2^{17}
(c) How large is the address space for I/O peripherals in bytes?	2^{16}
(d) If every peripheral must be allocated an address space of 2^{12} bytes, what is the largest number of peripherals that can be connected?	2^4


memory address bus (16 bits)
CPU
memory data bus (16 bits)
Memory
I/O data bus (8 bits)
I/O address bus (16 bits)
peripheral
peripheral
• • •
peripheral

QUESTION 2 [10 marks].

If you lost marks in this questions, I admit to disappointment. Conversion between bases is not a difficult exercise, it was predictable, it is well explained in the textbook and in the notes, and the numbers asked are small and easy. I think there was some carelessness and rush here, but I urge you to become a little more competent at these basic computations, as they will appear often and they will save you a lot of time.

(a) Assuming a 2’s complement representation, convert the following numbers from decimal to 8-bit binary and to its equivalent hexadecimal:

Decimal	Binary	Hexadecimal
35_{10}	0010 0011	23
-36_{10}	1101 1100	DC

(b) Given the 4-bit hexadecimal number, state its decimal equivalent according to the assumption of its representation listed in each heading:

Hexadecimal	*Unsigned*	*2’s complement*	*Signed magnitude*
E_{16}	14_{10}	-2_{10}	-6_{10}
4_{16}	4_{10}	4_{10}	4_{10}

QUESTION 3 [4 marks].

This question had been answered, in at least one form, in class, so I was very surprised by how many people did it incorrectly. I would like to emphasize that many people used the stack for the swapping (perfectly all right), but did not bother storing the final results back from the registers into the memory elements. The answer given was thus technically wrong, but I did not penalize it. People who used the XGDX instruction (a seemingly clever trick) forgot that both P and Q are only 1 byte each.

Give one or more 68HC11 instructions to swap the contents of the two bytes labelled P and Q, previously declared as:

```
          P   RMB  1
          Q   RMB  1
      LDAA    P
      LDAB    Q
      STAB    P
      STAA    Q
```

QUESTION 4 [6 marks].

Only the last item was tricky as it entailed remembering that BHS treats the numbers as “unsigned”. This was an easier version of the question from assignment 2.

Assume accumulator A contains the number -36_{10} and the 68HC11 instruction CMPA #10 has been executed. Which of the following branch instructions will result in the branch being taken towards “target”?

Branch	Yes/No
BEQ target	no
BPL target	no
BHS target	yes

QUESTION 5 [22 marks].

Most of you answered the first part of this question very well, but you should work on being precise when describing the “effect” of an instruction. You can use english instead of the technical notation, but you must be precise. About half the people showed confusion about what goes on the buses at what time. I am not entirely sure what to make of that, as we did examples in class, there were examples in the assignment and in the textbook. It is quite crucial that you understand the sequence of events, and what/how much data is being moved around.

(a) [10] Fill in the table below with the appropriate information about the instructions listed

Instruction	Addr. Mode	# bytes in code	# READ cycles	#WRITE cycles	What it does (use appropriate notation)
LDAA 0,X	IND	2	3	0	A <-- ((X) +0)
STAB $0123	EXT	3	3	1	($0123) <-- (B)
TXS	INH	1	1	0	SP <-- (X) -1
SUBA #12	IMM	2	2	0	A <-- (A) - 12

(b) [12] For each instruction, state exactly what happens in the “Fetch/Decode/Execute” cycle by indicating what goes on each bus and when. One instruction is given to you as an example with all the answers so you can see what is expected. You may not need all the spaces provided.

SUBA #12		what happens	Address Bus	Data Bus
	Step1	fetch instruction	content of PC = address of instruction	*going:* nothing *returning:* opcode
	Step 2	fetch operand	content of PC = address of operand	*going:* nothing *returning:* operand = 12
	Step 3	ALU does subtraction of A - 12, result in A	nothing	nothing
	Step 4	______________	_______________	_______________

LDAA 0,X		**what happens**	**Address Bus**	**Data Bus**
	Step1	fetch instruction	content of PC = address of instruction	*going:* nothing *returning:* opcode
	Step 2	fetch operand	content of PC = address of operand	*going:* nothing *returning:* offset = 0
	Step 3	compute effective address: (X) + 0 offset	_______________	_______________
	Step 4	fetch content of memory at effective address (X)+0	effective address calculated	*going:* nothing *returning:* memory content at effective address, i.e. at (X) +0

STAB $0123		**what happens**	**Address Bus**	**Data Bus**
	Step1	fetch instruction	content of PC = address of instruction	*going:* nothing *returning:* opcode
	Step 2	fetch operand	content of PC = address of operand	*going:* nothing *returning:* high byte of operand=$01
	Step 3	fetch operand	content of PC = address of operand	*going:* nothing *returning:* low byte of operand=$23
	Step 4	store content of B at address $0123	address $0123	*going:* content of B *returning:* nothing

QUESTION 6 [24 marks].
This questions was very well done by most people - congratulations! Understanding how the stack works is truly fundamental and not just at the assembly language level.

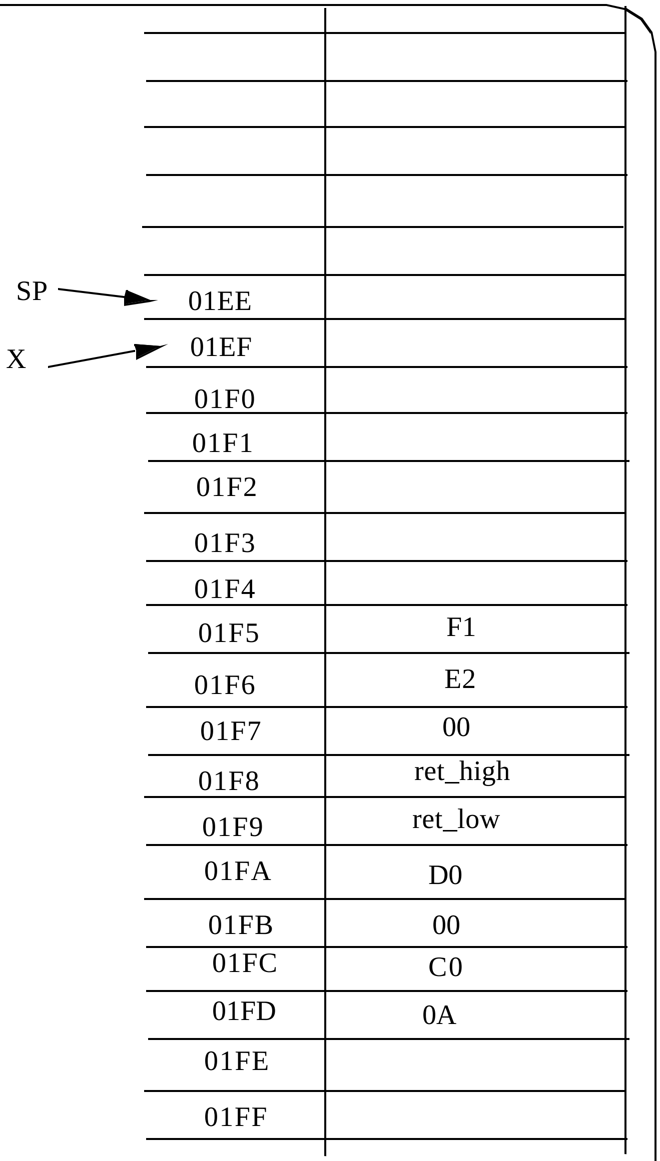
Assume the registers have been initialized as shown in the table below.

Draw the stack frame, enter the value of each stack slot (if it is known), and state the value of the register listed at the end of the given 68HC11 instruction sequence (write in the table).

Also write the precise address of each stack slot on the left end side of the stack picture.

Register	Initial	At End	work space (not marked)
A	$12	$01C0	
B	$20	$00	
X	$FFFF	$01EF	
Y	$F1E2	$F1E2	
D	$1220	$C000	
SP	$01FF	$01EE	

```
      LDS   #$01FF
      DES
      DES
      CLRB
      LDAA  #10
      PSHA
      LDAA  #$C0
      PSHA
      LDX   #$D000
      PSHX
      BSR   SUB1
      ........
      ........
SUB1  PSHB
      PSHY
      TSX
      XGDX
      SUBD  #6
      XGDX
      TXS
      ........
```



01EE	
01EF	
01F0	
01F1	
01F2	
01F3	
01F4	
01F5	F1
01F6	E2
01F7	00
01F8	ret_high
01F9	ret_low
01FA	D0
01FB	00
01FC	C0
01FD	0A
01FE	
01FF	

QUESTION 7 [20 marks].

Many people did not complete the whole question but did reasonably well in parts (a), (b) and (c). However, let me emphasize once again that putting comments and stating what a subroutine or program do implies giving meaningful documentation. A comment which simply paraphrases the assembly language instructions is not a useful or even valid piece of documentation. You had to show that you understood what the function of the program and subroutine overall are, and what is the structure of the algorithm used in the code. The details that one register is incremented or decremented is not what was expected. However, most answers were quite acceptable. One of the main items which was rewarded/penalized is whether you showed clearly that you had understood the difference between loading the address of a string or loading its content.

Consider the program below which manipulates three strings in some fashion using the subroutine “mystery”. Parameters are being passed through registers. You have 4 tasks:

(a) [4] place appropriate comments besides the code

(b) [4] state what the subroutine mystery does

Given 2 strings, whose addresses are in registers X and Y respectively, the subroutine copies the string pointed to by X to the area pointed to by Y. The strings are delimited by “0”.

(c) [4] state what the program does overall

The program concatenates 2 strings by copying them, one after the other, into another 3rd string, placing the “0” delimiter at the end.

(d) [12] rewrite using the stack for parameters by rewriting only the necessary lines or inserting some (use the numbering of lines given by the Assembler to be clear). For example, if inserting lines between 13 and 14, list them as 13.1, 13.2, etc.

One solution:

```
13.1        PSH   X
14.1        PSH   Y
..............................
15.1        PUL   X
15.2        PUL   X
...............................
16.1        PSH   X
16.2        PSH   Y
................................
17.1        PUL   X
17.2        PUL   Y
...............................
23.1        TSX
23.2        LDY   5,X
23.3        LDX   7,X
```

```
AS11M 01.05  Sun Feb 11, 2001 14:03  midterm1.lst

1           ;(a) state what the program does overall
2           ;(b) state what the subroutinemystery does
3           ;(c) place appropriate comments
4           ;(d) rewrite using the stack for parameters
5
6             ORG  $0001
7   string1  FCC  "xxxxxxxx"     ;string 1
8             FCB  0               ;end string 1
9   string2  FCC  "yyyyyyyy”     ;string 2
10            FCB  0               ;end string 2
11  string3  RMB  80
```

```
          ORG  $C000
          LDX  #string1  X <-- address of string 1
          LDY  #string3  Y <-- address of string 2
          JSR  mystery   call mystery to copy string 1 to 2
          LDX  #string2  X <-- address of string 1
          JSR  mystery   call mystery to copy string 1 to 2
          LDAB #0        B <-- 0 (end of string flag)
          STAB 0,Y       store end of string in string 3
          STOP
     mystery            subroutine “mystery”
                         in parameters: X contains address of
                         first string, Y address of second string
            PSHX        save registers X and B
            PSHB
     top    LDAB 0,X    B <-- element if 1st string
            BEQ  exit   if element=0, end of 1st string
            STAB 0,Y    copy element of 1st string in 2nd string
            INX         increment pointers to both strings
            INY         towards next elements
            BRA  top    repeat the loop till end of string 1
     exit   PULB        clear stack and restore registers
            PULX
            RTS
            END
```