UNIVERSITY OF VICTORIA
EXAMINATIONS DECEMBER 2001

C SC 230 Computer Architecture and Assembly Language

NAME (print) _____ REG NO. _____

SIGNATURE _____ DURATION: 3 hours

INSTRUCTOR D. MICHAEL MILLER

TO BE ANSWERED ON THIS EXAMINATION PAPER.

STUDENTS MUST COUNT THE NUMBER OF PAGES IN THIS EXAMINATION PAPER BEFORE BEGINNING TO WRITE, AND REPORT ANY DISCREPANCY IMMEDIATELY TO THE INVIGILATOR.

THIS EXAMINATION HAS **TEN** PAGES PLUS THIS COVER PAGE.

ATTEMPT EVERY QUESTION. ANSWER IN THE SPACES PROVIDED (YOU DO NOT NECESSARILY HAVE TO USE ALL LINES PROVIDED AND MAY USE OTHER AREAS ON THE **FRONTS** OF THE PAGES IF NECESSARY). USE THE BACKS OF THE PAGES FOR ROUGH WORK.

THIS IS A CLOSED BOOK EXAMINATION. **NO COURSE NOTES, BOOKS OR CALCULATORS, ARE PERMITTED.**

YOU **ARE** PERMITTED TO USE THE INFORMATION SHEETS DISTRIBUTED WITH THE EXAM.

QUESTION	MAX. MARK	STUDENT'S MARK
1	20	
2	5	
3	10	
4	8	
5	5	
6	10	
7	22	
8	20	
TOTAL	100	

1. (20 marks) Indicate whether each of the following statements is true or false by circling the appropriate response. **MARKING:** +1 for a correct answer , -0.5 for an incorrect answer, 0 if neither True of False is circled.

The 8-bit two's complement representation of -15_{10} is 11110000_2.	True	False
Two's complement representation has different representations for +0 and –0.	True	False
In two's complement addition, overflow can only occur when adding two positive or two negative numbers.	True	False
The stored exponent for the IEEE single precision floating point representations for 17.125 is 130 (in decimal).	True	False
Single bit parity allows for the detection and correction of only single bit errors.	True	False
An arithmetic shift always preserves the sign of the 2's complement value being shifted.	True	False
BCD arithmetic requires a decimal adjust instruction (DAA) to be executed after each addition.	True	False
On the 6811, the external address and data buses are synchronous.	True	False
A processor must have a stack pointer register in order to support a jump to subroutine instruction.	True	False
Subroutine parameters can not be used if the program code is in ROM.	True	False
Unless explicitly forbidden by the user program, nested maskable interrupts are allowed on the 6811.	True	False
Extended addressing on the 6811 means the address can exceed 16 bits in width.	True	False
The data bus to main memory must have the same number of bits as the word size.	True	False
On the 6811, on-processor memory can share addresses with off processor IO devices.	True	False
Polling should never be used if an interrupt can be used instead.	True	False
The 6811 is a RISC design.	True	False
The PENTIUM is a CISC design.	True	False
The PowerPC uses a stack to implement subroutine calls.	True	False
The PowerPC and the PENTIUM both support out of order instruction execution.	True	False
The PENTIUM II uses SIMD instructions to implement MMX.	True	False

2. (5 marks) The 6811 instruction `STA 10,Y` requires five machine cycles. Briefly explain what is done on each of those cycles:

   i) _____

   ii) _____

   iii) _____

   iv) _____

   v) _____

3. Answer each of the following in the space provided:

   (a) (2 marks) What is the functional difference between the RTS and RTI instructions on the 6811 processor?

   _____

   _____

   _____

   _____

   (b) (5 marks) Number the following steps from 1 to 5 in the order they are performed in processing an interrupt on the 6811 using the interrupt jump table technique

   ___ load the PC with the value from the appropriate interrupt vector

   ___ recognize the interrupt event and set the event flag

   ___ push the processor registers onto the stack

   ___ execute the first instruction of the interrupt handling routine

   ___ execute the appropriate jump instruction in the jump table

   (c) (3 marks) Explain how a compiler helps with branch prediction on the PowerPC.

   _____

   _____

   _____

4. Answer each of the following in the space provided:

   (a) (4 marks) What are the four basic stages in an instruction pipeline:

   _____

   _____

   _____

   _____

   (b) (2 marks) Why does a pipeline help improve instruction throughput on a processor?

   _____

   (c) (2 marks) What does it mean to say a processor is *superscalar*?

   _____

   _____

5. (5 marks) Consider the following segment of a 6811 program

```
AS11M 01.05  Thu Nov 29, 2001 21:40  test.lst

0001 dfff                        SBASE     EQU   $DFFF
0002 d000                                  ORG   $D000
0003 d000 48 41 52 52 59 20      SOURCE    FCC   "HARRY POTTER"
          50 4f 54 54 45 52
0004 d00c 00                               FCB   $00
0005 d00d                        DEST      RMB   80
0006 c000                                  ORG   $C000
0007 c000 8e df ff                         LDS   #SBASE
0008 c003 ce d0 00                         LDX   #SOURCE
0009 c006 3c                               PSHX
0010 c007 ce d0 0d                         LDX   #DEST
0011 c00a 3c                               PSHX
0012 c00b bd c0 13                         JSR   STRCPY
0013 c00e 31                               INS
0014 c00f 31                               INS
0015 c010 31                               INS
0016 c011 31                               INS
0017 c012 cf                               STOP
0018
0019 c013 36                     STRCPY    PSHA
0020 c014 37                               PSHB
0021 c015 3c                               PSHX
0022 c016 34                               DES
0023 c017 30                               TSX
```

Address (in hexadecimal)	*Contents*
_____	____
_____	____
_____	____
_____	____
_____	____
_____	____
_____	____
_____	____
_____	____
_____	____
_____	____
_____	____
_____	____
_____	____

Show the stack contents after the TSX is executed showing each BYTE on the stack and its location in hexadecimal. ***You may not need all the spaces provided.***

6. (10 marks) You are to write a 6811 assembly language function called POWER2 that accepts a byte as one parameter on the stack that is a value from 0 to 15 (you do not have to check that the value is in range). Your function should return with ACCD equal to 2^v where v is the value of the parameter provided. Your function should return with IX and IY unaltered. ***Include comments to explain what your code is doing.***

   For example, the function call

```
LDAA #7
PSHA
JSR  POWER2
```

   would return with decimal 128 in ACCD i.e. 0000000010000000_2. The call

```
LDAA #0
PSHA
JSR  POWER2
```

   would return with 1 in ACCD. ***You may not need all the space provided.***

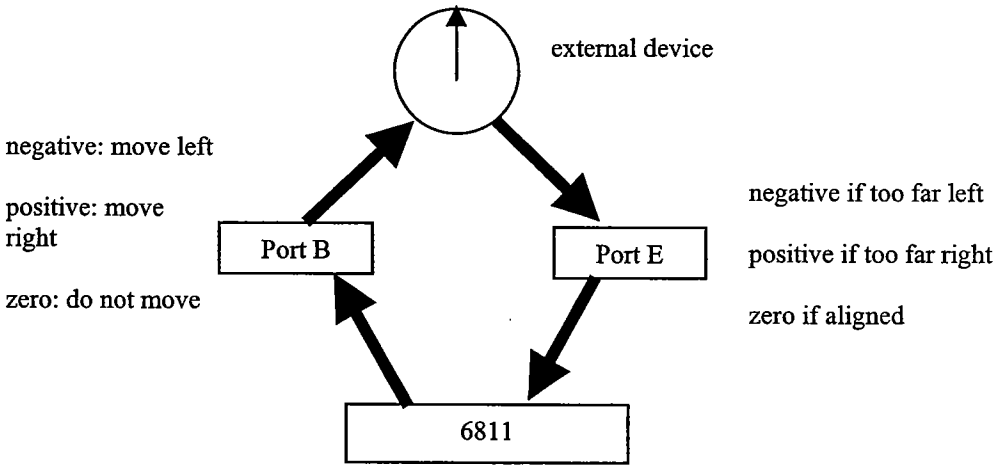
```
POWER2 _____
```

7. (22 marks) You may answer this question using C or 6811 assembly language. If you use C, you may include the "hc11.h" file listed on the handout. ***Assume the processor has a 2MHZ clock.***

   You are to write a **complete** program that controls the alignment of an external device interfaced to a 6811. The device constantly reports its position by providing a signed 8-bit number as input on Port E. A negative value means the device is positioned too far left, a positive number means it is positioned too far right. A value of 0 means the device is aligned.

   Your program moves the device by writing a signed 8-bit number to Port B. A negative value moves the device to the left, a positive value moves the device to the right, and a 0 means don't move the device. (See the picture below.)

   Your program is to read the value from Port E, and send half (integer division) the magnitude of the value read as the output control value on Port B. Make sure you set the correct sign for the output value – that is make sure if the device is too far left you are moving it to the right and *vice versa*. Your program should sample Port E **about** every second. ***Use TCNT overflow and polling for timing.***



You may not need all the space provided. ***Include comments.***

(continue on next page)

(question 7 continued)

8. (20 marks) **ATTEMPT EITHER 8A or 8B BUT NOT BOTH.** 8B is on page 9. **INDICATE WHICH ANSWER IS TO BE MARKED BY CIRCLING:** 8A 8B

8A: Complete the following stop watch program so it behaves as described. Note it is not the same as those you did for the assignments or the project. The routines ZERO and NDISPLAY are not shown for brevity. You do **not** have to write them. ***Assume the processor has a 4MHz clock***.

```
; This is a stop watch program that times in 1/100's of a second.
; The watch is in one of three states:
;     state 0 stopped with the time at 0 (initial state)
;     state 1 running
;     state 2 stopped with the time held at the time it was stopped
;
; The watch starts in state 0.  A rising edge on IC2 takes the watch
; to the next state in order 0 -> 1 -> 2 -> 0 etc.  So the first IC2
; event starts the watch.  A second one stops it, and a third one
; resets it to 0.
; OC2 interrupts are used for timing.

SBASE    EQU   $01FF
REGBASE  EQU   $1000
TCNT     EQU   $0E
TIC2     EQU   $12
TOC2     EQU   $18
TCTL2    EQU   $21
TMSK1    EQU   $22
TMSK2    EQU   $24
TFLG1    EQU   $23
IC2      EQU   $02
OC2      EQU   $40

; Timing control so watch counts in hundredths of a second
SLICE    EQU   _______
TIMECNT  EQU   _______

; Global variables
COUNT    RMB   1      / OC2 INTERRUPT COUNT
TIME     RMB   2      / TIME IN 0.01 SECS
STATE    RMB   1      / WATCH STATE

; Interrupt jump table entries

     ____________________
     ____________________
     ____________________
     ____________________

         ORG   $C000
START    LDS   #SBASE
         LDX   #REGBASE
         CLR   STATE     / WATCH IS INIT STOPPED
         JSR   ZERO      / AND ZEROED
         LDAA  _______   / SET IC2 EDGE TYPE TO RISING
         STAA  _______
```

(*continued on next page*)

(question 6 continued)

```
          LDAA  ________    / ENABLE IC2 INTERRUPTS
          STAA  ________
          ____              / ENABLE INTERRUPTS

LOOP      LDY   TIME       / DISPLAY TIME
          PSHY
          JSR   NDISPLAY
          INS
          INS
          BRA   LOOP
;=============================================================
; IC2 INTERRUPT

IC2       LDX   #REGBASE
          LDAA  ________      / CLEAR IC2 EVENT FLAG
          STAA  ________
          INC   STATE         / ADVANCE STATE
          LDAA  STATE
          CMPA  #1
          BEQ   IC21
          CMPA  #2
          BEQ   IC22
          CLR   STATE         / -> STATE 0 SO CLEAR TIMER
          JSR   ZERO
          RTI
IC21      LDD   ________      / -> STATE 1 SO SET TOC2
          ADDD  ________
          STD   TOC2,X
          BSET  ____________  / ENABLE OC2 INTERRUPT
          CLR   COUNT
          RTI
IC22      BCLR  ____________  / -> STATE 2 SO DISABLE OC2 INTERRUPT
          RTI
;=============================================================
; Time is kept by counting oc2 interrupts spaced SLICE cycles
; apart.  It takes TIMECNT slices to make 1/100 of a second.
; This handler affects the global variables COUNT and TIME.

OC2       ________________          continue below if needed
          ________________
          ________________          ________________
          ________________          ________________
          ________________          ________________
          ________________          ________________
          ________________          ________________
          ________________          ________________
          ________________          ________________
          ________________          ________________
          ________________          ________________
          ________________          ________________
```

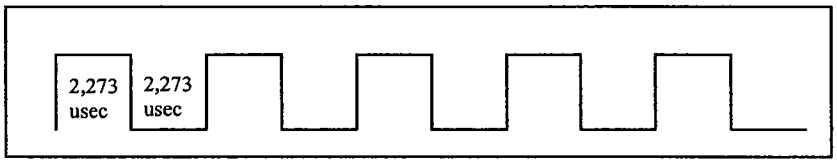
8B (Note you are to answer 8A or 8B but not both.) You may answer this question using C or 6811 assembly language. If you use C, you may include the "hc11.h" file listed on the handout. ***Assume the processor has a 1MHZ clock.***

Write a complete C or 6811 assembly language program that behaves as follows:

(a) After appropriate initialization, a **RISING** edge on input capture pin 1 (IC1), detected by an interrupt, causes the program to produce a 440 HZ. square wave on bit 6 of Port A.

(b) A **FALLING** edge on IC1 stops the production of the square wave as quickly as possible i.e. the program does not wait to complete the current cycle.

A 440 HZ square wave is a repeating signal for which each cycle consists of high for 2,273 usec. and then low for 2,273 usec. as shown below:



Include comments to explain what your code is doing.

***** End of Examination *****

Happy Holidays

M6811 INSTRUCTION SET

Registers:
A and B are 8-bit accumulators, D is the concatenation of A and B with A the most significant byte. IX and IY are unsigned 16 bit index registers. S is an unsigned 16-bit stack pointer, PC is an unsigned 16-bit program counter. CCR is the condition code register with flags SXHINZVC with C the least significant bit.

Addressing Modes :

Immediate:	IMM	data value is included in the instruction
Direct	DIR	operand is the page 0 address of the data value
Extended	EXT	operand is the 16-bit address of the data value
Indexed	IND	operand is an 8-bit unsigned offset which is added to the value from IX or IY to determine the address of the data value
Inherent	INH	opcode specifies registers
Relative	REL	destination of the branch is specified relative to the address in the PC

Instructions :
The instructions are shown in groups with each group preceded by the allowed addressing modes and the flags affected for the group. The addressing modes are in italics. The flags affected are underlined. none means no flags are affected..

Load: *IMM,DIR,EXT,IND* NZV=0 LDAA, LDAB, LDD, LDS, LDX, LDY

Store: *DIR,EXT,IND* NZV=0 STAA, STAB, STD, STS, STX, STY

Transfer: *INH* NZ TAB, TBA
INH none TSX, TXS, TSY, TYS

Exchange: *INH* none XGDX, XGDY

Stack: *INH* none DES, INS, PSHA, PSHB, PSHX, PSHY, PULA, PULB, PULX, PULY

Arithmetic: *INH* HNZVC ABA
IMM,DIR,EXT,IND HNZVC ADDA, ADDB, ADCA, ADCB
INH none ABX, ABY, INS, DES
INH NZVC SBA *note:* A←(A)-(B)
IMM,DIR,EXT,IND NZVC ADDD, SUBA, SUBB, SUBD, SBCA, SBCB
INH NZV=?C DAA
EXT,IND NZV DEC, INC
INH NZV DECA, DECB, INCA, INCB
INH Z DEX, DEY, INX, INY, *INH* NZVC NEGA, NEGB
EXT,IND NZVC NEG
INH ZVC FDIV *fractional:* (D)/(IX) IX←quotient D←remainder
INH ZV=0C IDIV *integer:* (D)/(IX) IX←quotient D←remainder
INH C MUL *multiplication:* D←(A) * (B)

Logical: *IMM,DIR,EXT,IND* NZV=0 ANDA, ANDB, ORAA, ORAB, EORA, EORB
INH NZV=0C=1 COMA, COMB
EXT,IND NZV=0C=1 COM

Shift & Rotate: *INH* NZVC LSLA, LSLB, LSLD=ASLD, ASLA, ASLB, ASLD=LSLD, ASRA, ASRB, RORA, RORB, ROLA, ROLB
INH N=0ZVC LSRA, LSRB, LSRD
EXT,IND NZVC LSL, ASL,ASR,ROR,ROL, *EXT,IND* N=0ZVC LSR
note: operation of C flag is dictated by the type of shift or rotate.

Decision Making:

No Flag	Carry Flag	Zero Flag	Sign(N) Flag
BRA	BCC, BCS	BEQ, BNE	BMI, BPL
JMP			
Overflow Flag	**2's Complement**	**Unsigned**	**Bit Test**
BVS, BVC	BGE, BGT	BHI, BHS	BRCLR
	BLE, BLT	BLO, BLS	BRSET

B** instructions use *REL*, BRCLR & BRSET use *DIR* or *IND*, JMP uses *EXT* or *IND*
None affect any flags .

Test / Compare:
IMM,DIR,EXT,IND NZV=0 BITA, BITB
INH NZVC CBA *note:* flags set based on (A) - (B)
IMM,DIR,EXT,IND NZVC CMPA, CMPB, CPD, CPX, CPY
INH NZV=0C=0 TSTA, TSTB
EXT,IND NZV=0C=0 TST

Subroutine Linkage:
REL none BSR, *DIR,EXT,IND* none JSR, *INH* none RTS

Interrupt Handling: *INH* I CLI, SEI
INH I=1 SWI
INH none WAI
INH SHINZVC RTI

Setting / Clearing:
DIR,IND NZV=0 BCLR, BSET,
INH Z=1 N=V=C=0 CLRA, CLRB
EXT,IND Z=1 N=V=C=0 CLR

Condition Code
INH SHINZVC TAP
INH none TPA,
INH C CLC, SEC
INH V CLV, SEV

Miscellaneous: *INH* None BRN, NOP, STOP

M6811 Timer Section

Input Captures The 6811 timer section has three input capture pins which can be used to capture an external event. The event can be programmed to be a rising edge, a falling edge or either edge. When an input event occurs the value of the free-running timer TCNT is copied into a time of capture(TIC) register.

Input capture specifics:

Input Capture	Port Location	Int. Vector	Jump Table LOcation	Time of Input Capture (TIC)
1	Port A, Bit 2	$FFEE-$FFEF	$00E8	$1010-$1011
2	Port A, Bit 1	$FFEC-$FFED	$00E5	$1012-$1013
3	Port A, Bit 0	$FFEA-$FFEB	$00E2	$1014-$1015

The type of event, the input capture flag and the interrupt enable are contained in three registers

Name	Addr.	B7	B6	B5	B4	B3	B2	B1	B0
TCTL2	$1021	0	0	E1B	E1A	E2B	E2A	E3B	E3A
TMSK1	$1022						IC1I	IC2I	IC3I
TFLG1	$1023						IC1F	IC2F	IC3F

ICnI is the interrupt enable flag for input capture n and ICnF is the event flag for input capture n.

The edge sensitivity is programmed as follows:

EnB	EnA		EnB	EnA	
0	0	Capture disabled	1	0	Capture on falling edge
0	1	Capture on rising edge	1	1	Capture on either edge

Output Compares The 6811 has 5 output compares (OC), but you should avoid using OC1 as it is a multi-function pin with several unique features. The program loads TOC register with a value and when TCNT reaches the specified value, the output compare event happens. The particular event is programmable and an interrupt can be generated.

Output compare specifics:

Output Compare	Port Location	Int. Vector	Jump Table Location	Time of Output Compare (TOC)
1	Port A, Bit 7	$FFE8-$FFE9	$00DF	$1016-$1017
2	Port A, Pin 6	$FFE6-$FFE7	$00DC	$1018-$1019
3	Port A, Pin 5	$FFE4-$FFE5	$00D9	$101A-$101B
4	Port A, Pin 4	$FFE2-$FFE3	$00D6	$101C-$101D
5	Port A, Pin 3	$FFE0-$FFE1	$00D3	$101E-$101F

The type of event, the input capture flag and the interrupt enable are contained in three registers

Name	Addr.	B7	B6	B5	B4	B3	B2	B1	B0
TCTL1	$1020	OM2	OL2	OM3	OL3	OM4	OL4	OM5	OL5
TMSK1	$1022	OC1I	OC2I	OC3I	OC4I	OC5I			
TFLG1	$1023	OC1F	OC2F	OC3F	OC4F	OC5F			

The output event is programmed as follows:

OMn	OLn		OMn	OLn	
0	0	Pin is not affected (OC1 may be)	1	0	Clear OCn to 0
0	1	Toggle OCn	1	1	Set OCn to 1

PORT Addresses: A $1000; B $1004; C $1003; D $1008; E $100A

Other Addresses: DDRC $1007; DDRD $1009; TCNT $100E,$100F

```
/************************************************/
/*                                              */
/* C SC 230 Course Software                     */
/* 68HC11 standard register section definitions */
/*                                              */
/************************************************/

#define _IO_BASE  0x1000
#define PORTA     *(unsigned char volatile *)(_IO_BASE + 0x00)
#define PIOC      *(unsigned char volatile *)(_IO_BASE + 0x02)
#define PORTC     *(unsigned char volatile *)(_IO_BASE + 0x03)
#define PORTB     *(unsigned char volatile *)(_IO_BASE + 0x04)
#define PORTCL    *(unsigned char volatile *)(_IO_BASE + 0x05)
#define DDRC      *(unsigned char volatile *)(_IO_BASE + 0x07)
#define PORTD     *(unsigned char volatile *)(_IO_BASE + 0x08)
#define DDRD      *(unsigned char volatile *)(_IO_BASE + 0x09)
#define PORTE     *(unsigned char volatile *)(_IO_BASE + 0x0A)
#define CFORC     *(unsigned char volatile *)(_IO_BASE + 0x0B)
#define OC1M      *(unsigned char volatile *)(_IO_BASE + 0x0C)
#define OC1D      *(unsigned char volatile *)(_IO_BASE + 0x0D)
#define TCNT      *(unsigned short volatile *)(_IO_BASE + 0x0E)
#define TIC1      *(unsigned short volatile *)(_IO_BASE + 0x10)
#define TIC2      *(unsigned short volatile *)(_IO_BASE + 0x12)
#define TIC3      *(unsigned short volatile *)(_IO_BASE + 0x14)
#define TOC1      *(unsigned short volatile *)(_IO_BASE + 0x16)
#define TOC2      *(unsigned short volatile *)(_IO_BASE + 0x18)
#define TOC3      *(unsigned short volatile *)(_IO_BASE + 0x1A)
#define TOC4      *(unsigned short volatile *)(_IO_BASE + 0x1C)
#define TOC5      *(unsigned short volatile *)(_IO_BASE + 0x1E)
#define TCTL1     *(unsigned char volatile *)(_IO_BASE + 0x20)
#define TCTL2     *(unsigned char volatile *)(_IO_BASE + 0x21)
#define TMSK1     *(unsigned char volatile *)(_IO_BASE + 0x22)
#define TFLG1     *(unsigned char volatile *)(_IO_BASE + 0x23)
#define TMSK2     *(unsigned char volatile *)(_IO_BASE + 0x24)
#define TFLG2     *(unsigned char volatile *)(_IO_BASE + 0x25)
#define PACTL     *(unsigned char volatile *)(_IO_BASE + 0x26)
#define PACNT     *(unsigned char volatile *)(_IO_BASE + 0x27)
#define SPCR      *(unsigned char volatile *)(_IO_BASE + 0x28)
#define SPSR      *(unsigned char volatile *)(_IO_BASE + 0x29)
#define SPDR      *(unsigned char volatile *)(_IO_BASE + 0x2A)
#define BAUD      *(unsigned char volatile *)(_IO_BASE + 0x2B)
#define SCCR1     *(unsigned char volatile *)(_IO_BASE + 0x2C)
#define SCCR2     *(unsigned char volatile *)(_IO_BASE + 0x2D)
#define SCSR      *(unsigned char volatile *)(_IO_BASE + 0x2E)
#define SCDR      *(unsigned char volatile *)(_IO_BASE + 0x2F)
#define ADCTL     *(unsigned char volatile *)(_IO_BASE + 0x30)
#define ADR1      *(unsigned char volatile *)(_IO_BASE + 0x31)
#define ADR2      *(unsigned char volatile *)(_IO_BASE + 0x32)
#define ADR3      *(unsigned char volatile *)(_IO_BASE + 0x33)
#define ADR4      *(unsigned char volatile *)(_IO_BASE + 0x34)
#define OPTION    *(unsigned char volatile *)(_IO_BASE + 0x39)
#define COPRST    *(unsigned char volatile *)(_IO_BASE + 0x3A)
#define PPROG     *(unsigned char volatile *)(_IO_BASE + 0x3B)
#define HPRIO     *(unsigned char volatile *)(_IO_BASE + 0x3C)
#define INIT      *(unsigned char volatile *)(_IO_BASE + 0x3D)
#define TEST1     *(unsigned char volatile *)(_IO_BASE + 0x3E)
#define CONFIG    *(unsigned char volatile *)(_IO_BASE + 0x3F)
```