CS224 – Spring 2019 - Lab #7 (Version 2, May 11, 2019 13:02 am)

Programming PIC32 Microcontroller

Dates: The same for all sections. Wednesday May 15. 13:30. You do the work at home, no need to come to the lab. You have to do the work individually: It is NOT A TEAM WORK.

Purpose:
The new version of the lab assignment only involves C to MIPS assembly language conversion. Purpose of the practice is understanding the fundamental principles of PIC32 programming that involves I/O and bit operations on this microcontroller.

Summary
Introduction to PIC32 programming.

DUE DATE: SAME FOR ALL SECTIONS:

1. Please bring and drop your work into the box(es) provided in front of TA office room number EA-407 by 13:30 on Wednesday May 15.
   YES paper submission this time too.

2. Please **upload your work** to the Unilica by 13:30 on Wednesday May 15. Use filename **StudentID_FirstName_LastName_SecNo_LabNo.txt** Only a NOTEPAD FILE (txt file) is accepted.

You are required to read the instructions provided on this document. No excuses will be accepted.

YES paper submission this time too. YES paper submission this time too. YES paper submission this time too. YES paper submission this time too. YES paper submission this time too.

No late submission will be accepted. **No late submission will be accepted**. **No late submission will be accepted**. **No late submission will be accepted**. **No late submission will be accepted**.

Upload txt. Upload txt. Upload txt. Upload txt. Upload txt. Upload txt. Upload txt. Upload txt. Upload txt. Upload txt. Upload txt. Upload txt. Upload txt.

C to MIPS Assembly Language Conversion (100 points)

At the top of your submission provide the following information.

CS224
Section No.: ...
Spring 2019
Lab No.:
Your Full Name/Bilkent ID:

Convert the C programs 1 & 2 into MIPS assembly language.

For register addresses of PIC32 please refer to the class handout or the PIC32 datasheets (see the Lab7 Unilica folder).

In the body make sure that you clearly indicate what is provided for what:

1. Clearly indicate the programs. At the beginning of each program provide your name and section in a comment line.
2. Use consistent and meaningful format in your MIPS code. Neat and understandable presentation is important during grading.
3. In your conversion from C to MIPS for each C statement first give the C statement as a MIPS comment line then provide the symbolic MIPS instruction(s) you generate for that C statement. Do this for each C statement so that your TA will be able to follow your presentation. If this is not provided 50 points will be taken off from your grade upfront.
4. If a C code line does not require any code in MIPS, like declarations etc., just provide the C code.
5. In your presentation delete the comments of the C code.

Program No. 1 (40 points)

```
void initspi(void) {
       char junk;
       SPI2CONbits.ON = 0; // disable SPI to reset any previous state
       junk = SPI2BUF; // read SPI buffer to clear the receive buffer
       SPI2BRG = 7;
       SPI2CONbits.MSTEN = 1;// enable master mode
       SPI2CONbits.CKE = 1; // set clock-to-data timing
       while(!SPI2CONbits.DONESSEN); // Check the SSEN bit
       SPI2CONbits.ON = 1; // turn SPI on
}
```

Program No. 2 (60 points)

```
// This code shows and rotates the pattern (10001000) right or stops based on the
input coming from the user. The pattern is to be shown on the LEDs.

int stop = 0;
int initial  = 0b01110111; //Initial pattern. Note that 0 means on, while 1 means off.
int right = 1;

void main(){
     TRISD = 0x0; // All bits of PORTD are output. ~0 means output~

// Three bits of PORTA are inputs but only one of them is used in this example as a
stop button, others are redundant. ~1 means input~

       TRISA = 0b111;

// From PORTD, outputs will be sent to LEDs. Make sure that you physically connected
them by looking at the Figure 1, in the directives document.
// Initial pattern is sent to the LEDs through PORTD.

     PORTD = initial;
     while(1){
              int lsb;
              int mask;
              // Stop button is the push-button which is labeled as 1 on the board.
              if(PORTABits.RA1 == 0){ // If stop button clicked
                   stop = !stop;
                   if(!stop){
// If process restarted, copy initial pattern into PORTD.
                        PORTD = initial;
                   }
              }
              if(!stop){
                    //Rotate right
                 lsb = PORTD & 0x1; // Extract least significant bit
                 mask = lsb << 7; // Least significant bit will be the msb of the
                                  // shifted pattern
                 PORTD = (PORTD >> 1) | mask; // Paste lsb to the leftmost bit the
                                              // right shifted portd
              } else {
                //Do not shift anything, that is, stop.
                PORTD = 0b11111111;
              }
              delay_ms(1000); // Wait 1 second.
     }
}
```

APPENDIX

HOW TO FIND MEMORY ADDRESSES OF REGISTERS AND HOW TO ACCESS THEIR BITS: AN EXAMPLE

Translate the following C code into MIPS assembly code, as the PIC32 compiler would based on the p32xxxx header file. You can refer to the Table 4-16 from the PIC32 datasheet. Use only real MIPS instructions.

```
int  readadc (void)  {
  AD1CON1bits.SAMP = 0 ;
  while (!AD1CON1bits.DONE) ;
  AD1CON1bits.SAMP = 1 ;
  AD1CON1bits.DONE = 0 ;
  return (ADC1BUF0) ;
}
```

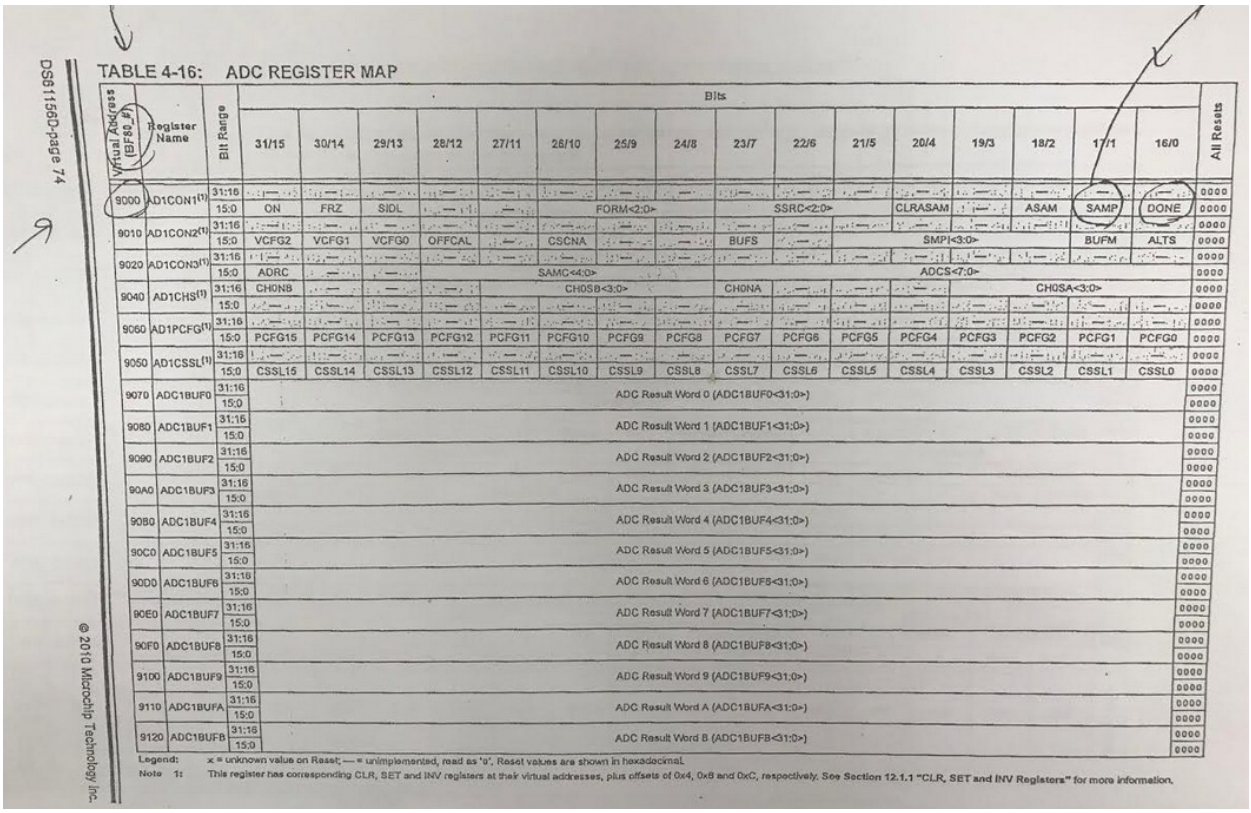
```
readadc:  lui $t0, 0xBF80          // load AD1CON1 address into $t0
          ori $t0, $t0, 0x9000[1]        // (0xBF809000 from Table 4-16)
          lw $t1, 0 ($t0)          // get AD1CON1 value into register $t1
          andi $t1, $t1, 0xFFFD    // AND mask to clear bit 1 (SAMP bit)
          sw $t1, 0 ($t0)          // store value back to AD1CON1
while:    lw $t1, 0 ($t0)          // get AD1CON1 value into register $t1
          andi $t2, $t1, 0x0001    // AND mask to clear all bits except bit 0
                                   //   (DONE bit)
          beq $t2, $0, while       // spin in while loop until DONE = 1
          ori $t1, $t1, 0x0002     // OR mask to set bit 1 (SAMP bit)
          andi $t1, $t1, 0xFFFE    // AND mask to clear bit 0 (DONE bit)
          sw $t1, 0 ($t0)          // store value back to AD1CON1
          lw $v0, 0x70 ($t0)       // load ADC1BUF0 into $v0
                                   // (address is 0xBF809070 in Table 4-16)
          jr $ra                   // return to callee
```

What is the address of ADCON1: Find AD1CON1 in the data sheets of PIC32 (see the documents folder in Unilica for the data sheets, or web). On that table take the number on the left upper corner (BF80) from the table contains AD1CON1 and take the number horizontally aligned with AD1CON1 (9000) and concatenate them, it becomes 0XBF809000. See the firts 3 lines of the MIPS code above.

How to access AD1CON1bits.SAMP: Bit names of AD1CON1 are horizontally available next to AD1CON1 on the right hand side in two narrow lines/rows. Lower line shows the bit positions 0 to 15 and the upper line shows the bit position 16 to 31/ Notice the notation

[1] In my class presentation I was thinking that there is a bug in this lui-ori approach. Actually ori instruction zerozextents the sign bit, so there is no bug. See the green card or see Figure 6.15 in the textbook.

31/15 to 16/0. Bit position of SAMP is 2. For clearing the SAMP bit see the 4th line of the MIPS code.



DS61156D-page 74

TABLE 4-16: ADC REGISTER MAP

Virtual Address (BF80_#)	Register Name	Bit Range	Bits 31/15	30/14	29/13	28/12	27/11	26/10	25/9	24/8	23/7	22/6	21/5	20/4	19/3	18/2	17/1	16/0	All Resets
9000	AD1CON1(1)	31:16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
		15:0	ON	FRZ	SIDL	—	—	FORM<2:0>			SSRC<2:0>			CLRASAM	—	ASAM	SAMP	DONE	0000
9010	AD1CON2(1)	31:16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
		15:0	VCFG2	VCFG1	VCFG0	OFFCAL	—	CSCNA	—	—	BUFS	—	SMPI<3:0>				BUFM	ALTS	0000
9020	AD1CON3(1)	31:16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
		15:0	ADRC	—	—	SAMC<4:0>					ADCS<7:0>								0000
9040	AD1CHS(1)	31:16	CH0NB	—	—	—	CH0SB<3:0>				CH0NA	—	—	—	CH0SA<3:0>				0000
		15:0	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
9060	AD1PCFG(1)	31:16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
		15:0	PCFG15	PCFG14	PCFG13	PCFG12	PCFG11	PCFG10	PCFG9	PCFG8	PCFG7	PCFG6	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0	0000
9050	AD1CSSL(1)	31:16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
		15:0	CSSL15	CSSL14	CSSL13	CSSL12	CSSL11	CSSL10	CSSL9	CSSL8	CSSL7	CSSL6	CSSL5	CSSL4	CSSL3	CSSL2	CSSL1	CSSL0	0000
9070	ADC1BUF0	31:16	ADC Result Word 0 (ADC1BUF0<31:0>)																0000
		15:0																	0000
9080	ADC1BUF1	31:16	ADC Result Word 1 (ADC1BUF1<31:0>)																0000
		15:0																	0000
9090	ADC1BUF2	31:16	ADC Result Word 2 (ADC1BUF2<31:0>)																0000
		15:0																	0000
90A0	ADC1BUF3	31:16	ADC Result Word 3 (ADC1BUF3<31:0>)																0000
		15:0																	0000
90B0	ADC1BUF4	31:16	ADC Result Word 4 (ADC1BUF4<31:0>)																0000
		15:0																	0000
90C0	ADC1BUF5	31:16	ADC Result Word 5 (ADC1BUF5<31:0>)																0000
		15:0																	0000
90D0	ADC1BUF6	31:16	ADC Result Word 6 (ADC1BUF6<31:0>)																0000
		15:0																	0000
90E0	ADC1BUF7	31:16	ADC Result Word 7 (ADC1BUF7<31:0>)																0000
		15:0																	0000
90F0	ADC1BUF8	31:16	ADC Result Word 8 (ADC1BUF8<31:0>)																0000
		15:0																	0000
9100	ADC1BUF9	31:16	ADC Result Word 9 (ADC1BUF9<31:0>)																0000
		15:0																	0000
9110	ADC1BUFA	31:16	ADC Result Word A (ADC1BUFA<31:0>)																0000
		15:0																	0000
9120	ADC1BUFB	31:16	ADC Result Word B (ADC1BUFB<31:0>)																0000
		15:0																	0000

Legend: x = unknown value on Reset; — = unimplemented, read as '0'. Reset values are shown in hexadecimal.

Note 1: This register has corresponding CLR, SET and INV registers at their virtual addresses, plus offsets of 0x4, 0x8 and 0xC, respectively. See Section 12.1.1 "CLR, SET and INV Registers" for more information.