Devdhar Kadiatar

ME 333
Homework 7

02/03/19

Chapter 3 :-

1. a). VA = 0x80000020 → cacheable memory address Data RAM

⇒ PA = 0x00000020

b). VA = 0xA0000020 → non-cacheable memory address Data RAM

⇒ PA = 0x00000020

c). VA = 0xBF800001 → non-cacheable memory address SFR

⇒ PA = 0x1F800001

d). VA = 0x9FC00111 → cacheable memory address boot flash

⇒ PA = 0x1FC00111

e). VA = 0x9D001000 → cacheable memory address program flash

⇒ PA = 0x1D001000

2. For NU32bootloaded.ld,

_RESET_ADDR = 0xBD001970

PTO →

3. a). The following bits can be used as input/output,

PORT B : bits 0-15
PORT C : bits 12-15
PORT D : bits 0-11
PORT E : bits 0-7
PORT F : bits 0, 1, 3, 4, 5
PORT G : bits 2, 3, 6, 7, 8, 9

RE0 corresponds to pin 60.

b). SFR INTCON :-

Unimplemented bits : 5, 6, 7, 11, 13, 14, 15, 17-31

Implemented bits : 0-4 (INTxEP), 8-10 (TPC<2:0>)
12 (MVEC), 16 (SSO)

4. simplePICmodify.c attached.

6. I2C3CON : Virtual address = 0xBF805000
Reset value = 0x00001000

TRISC : Virtual address = 0xBF886080
Reset value = 0x0000F000

PTO →

8. a).

```
and     a0, a0, 0
and     a1, a1, 0
la      t0, _main_entry
jr      t0
nop
```

.end startup

b). SFRs with the highest address (ascending),

C2FIFOCI31INV	at	BF88CB4C
DEVCFG3	at	BFC02FF0
DEVCFG2	at	BFC02FF4
DEVCFG1	at	BFC02FF8
DEVCFG0	at	BFC02FFC

c). For the data type __SPI2STATbits_t, 11 bit fields are defined namely,

SPIRBF	(1 bit)
SPITBF	(1 bit)
SPITBE	(1 bit)
SPIRBE	(1 bit)
SPIROV	(1 bit)
SRMT	(1 bit)
SPITUR	(1 bit)
SPIBUSY	(1 bit)
TXBUFELM	(5 bits)
RXBUFELM	(5 bits)
and, w	(32 bits)

From the Data sheet (SPI2 Register Map) we see that the bit fields in p32mx795f512h.h file are in accordance.

9. 1). TRISDSET = 0xC ; // TRISDSET = 0b1100

2). TRISDCLR = 0b100010 ;

3). TRISDINV = 0b10001 ;

7. The processor.o file contains the addresses of all the SFRs that the PIC32 can possibly use. This is why the file size is big. After linking with other object codes to produce the elf file, the hex file is finally produced which is usually less than 10 KB.

Chapter 4 :-

1. The following are meant to be used in other c files,

NU32_ReadUART3 and NU32_Startup(), NU32_WriteUART3

NU32_DESIRED_BAUD is private to NU32.c

2. a). investPIC.c attached

b). The main.c consists of the main function which calls the functions to be used. The function definations are given in helper.h along with other constants used. The descriptions of these functions are given in helper.c which also include helper.h.

c). The io.h file contains function definations of getUserInput and sendOutput, which is is descrited in io.c. io.h also contains the structure Investment and MAX_YEARS

The calculate.h file contains only the function defination of calculateGrowth, described in calculate.c.

Note :- calculate.c will have an #include "io.h" because calculateGrowth uses Investment which is defined in io.h.

4.
```
void    LCD_ClearLine (int ln)
{
        char str[16] = "                " // 16 space characters
        if (ln == 0)
        {
                LCD_Move (0, 0); //moves cursor to line 0
                LCD_WriteString (str); //clears line 0
        }
        else if (ln == 0)
        {
                LCD_Move (1, 0); //moves cursor to line 1
                LCD_WriteString (str); // clears line 1
        }
        else
        {
                printf("Invalid Input value \n");
        }
}
```

Chapter 5 :-

3. a). The following commands has the jump 'jal'

```
j3 = j1/j2 ;
f3 = f1+f2 ;
f3 = f1-f2 ;
f3 = f1*f2 ;
f3 = f1/f2 ;
```

→ assembly code for this line is,

```
9d003134 : 8fc40030    lw    a0, 48(s8)
9d003138 : 8fc50034    lw    a1, 52(s8)
9d00313c : 0f400dc4    jal   9d003710 <divsf3>
9d003140 : 00000000    nop
9d003144 : afc20058    sw    v0, 88(s8)
```

```
d3 = d1+d2 ;
d3 = d1-d2 ;
d3 = d1*d2 ;
d3 = d1/d2 ;
```

b). The following commands have the least assembly code (4 lines) (without a jump)

```
i3 = i1+i2 ;

i3 = i1-i2 ; →  9d002fe4 : 8fc30014   lw   v1, 20(s8)
                9d002fe8 : 8fc20018   lw   v0, 24(s8)
                9d002fec : 00621023   subu v0, v1, v0
                9d002ff0 : afc2004c   sw   v0, 76(s8)
```

With some character arithmetic, the assembly code has an extra andi line which turns all the bits in V0 into 1.

c).

	char	int	long long	float	long double
+	1.25(5)	1.0(4)	2.75(11)	1.25(5) J	2.0(8) J
−	1.25(5)	1.0(4)	2.75(11)	1.25(5) J	2.0(8) J
*	1.5(6)	1.25(5)	4.75(19)	1.25(5) J	2.0(8) J
/	1.75(7)	1.75(7)	3.0(12) J	1.25(5) J	2.0(8) J

d). Math subroutines :-

Subroutines	VA	memory (bytes)
.text.dp32mul	0x9d001e00	1208
.text.dp32subadd	0x9d0022b8	1072
.text.dp32mul	0x9d002b9c	812
.text.fpsubadd	0x9d003498	632
.text.fp32div	0x9d003710	560
.text.fp32mul	0x9d003940	444

4. & and | operators expands to 4 assembly lines while << and >> operator commands expand to 3 assembly lines.

<< and >> are a faster option than multiplying or dividing a 2^n number.

6. a). Using the core timer, elapsedticks = _CP0_GET_COUNT
elapsedns = elapsedticks *25;
we can get the time for the program to compile in nanoseconds.
Divide this value by 12.5 ns gives SYSCLK cycles for each of the commands.

b). f2 = cosf(f1);

```
9d002780: 8fc40010  lw  a0,16(s8)
9d002790: 0f400898  jal 9d002260 <cosf>
9d002794: 00000000  nop
9d002798: afc20020  sw  v0,32(s8)
```

d2 = cos(d1);

```
9d0027ac: 8fc40018  lw  a0,24(s8)
9d0027b0: 8fc5001c  lw  a1,28(s8)
9d0027b4: 0f400921  jal 9d002484 <__truncdfsf2>
9d0027b8: 00000000  nop
9d0027bc: 00402021  move a0,v0
```

```
9d0027c0 : 0f400898   jal 9d002260 <cosf>
9d0027c4 : 00000000   nop
9d0027c8 : 00402021   move a0, v0
9d0027cc : 0f400a0a   jal 9d002828 <.LFE0>
9d0027d0 : 00000000   nop
9d0027d4 : afc20028   sw v0, 40(s8)
9d0027d8 : afc3002c   sw v1, 44(s8)
```

Advantages :- long double calculates the value at a higher precision

Disadvantages :- using long double for cosine calculations are 3 times slower than using float.
It also contains 3 jumps within the assembly code.

c). Directory of libm.a :- (Full Path)

c:/program files (x86)/microchip/xc32/v2.15/pic32mx/lib\libm.a

10. Total memory reserved for stack is 131032 bytes.

$\therefore$ 131032 - 5000×4 = 111032 bytes

This is how much memory is avaible for the int array.

$\therefore$ Max interger array :- int a[27758]