Lecture 12 - Trace of A MARIE Program or Two

C Code and Operators (This also applies to C++)

These work in C, C++, Java, JavaScript (node).

Arithmetic Shift Left

ex1.c:

```
#include <stdio.h>

int main() {
    int x;
    x = 1;
    x = x << 1;        /* Same as multipy by 2 */
    printf ( "%d\n", x );
}
```

Arithmetic Shift Right

ex2.c:

```
#include <stdio.h>

int main() {
    int x;
    x = 15;
    x = x >> 1;        /* Same as devide by 2 */
    printf ( "%d\n", x );
}
```

Get a specific BIT using and

ex3.c

```
#include <stdio.h>

int main() {
    int x;
    x = 0x12;                   /* Hex value */
    x = x & 0x10;               /* Pick out just 1 bit */
```

```
    printf ( "0x%x (Expect 0x10)\n", x );    /* Print in Hex */
    x = 0x402;                    /* Hex value */
    x = x & 0x10;                /* Pick out just 1 bit */
    printf ( "0x%x (Expect 0x0)\n", x );    /* Print in Hex */
}
```

Formatted a little better.

ex4.c:

```
#include <stdio.h>

int main() {
    int x;
    x = 0x12;                    /* Hex value */
    x = x & 0x10;                /* Pick out just 1 bit */
    printf ( "0x%04x (Expect 0x0010)\n", x );    /* Print in Hex */
    x = 0x402;                    /* Hex value */
    x = x & 0x10;                /* Pick out just 1 bit */
    printf ( "0x%04x (Expect 0x0000)\n", x );    /* Print in Hex */
}
```

Get a bit and shift it to least significant postion

ex5.c:

```
#include <stdio.h>

int main() {
    int x;
    x = 0xE023;                    /* Hex value */
    x = x & 0xF000;                /* Pick out just 4 bit */
    x = x >> 12;                /* Shift it over */
    printf ( "0x%04x (Expect 0x000E)\n", x );    /* Print in Hex */
}
```

Set some bits in a value

ex6.c:

```
#include <stdio.h>

int main() {
```

```
    int x;
    x = 0xE023;                   /* Hex value */
    x = x | 0x0100;               /* Set a single bit */
    printf ( "0x%04x (Expect 0xE123)\n", x );    /* Print in Hex */
}
```

Toggle some bits in a value (Use XOR)

ex8.c:

```
#include <stdio.h>

int main() {
    int x;
    x = 0x23;                   /* Hex value */
    x = x ^ 0xF0;               /* Toggle with XOR */
    printf ( "0x%02x (Expect 0xD3)\n", x );    /* Print in Hex */
}
```

Split up IR value into the OP and the HAND.

ex7.c:

```
#include <stdio.h>

int main() {
    int x, op, hand;
    x = 0x4022;                   /* Add instruction from address 22 hex */
    op = x & 0xF000;
    op >>= 12;                   /* a >>= b  same as a = a >> b */
    hand = x & 0xFFF;
    printf ( "Op = 0x%x, hand = 0x%03x \n", op, hand );
}
```

Compliment 1s

ex9.c:

```
#include <stdio.h>

int main() {
    unsigned int x;            /* Note "unsigned" int */
```

```
    x = 0x23;                   /* Hex value */
    x = ~x;                       /* Get the 1s Compliment */
    x &= 0xFFFF;                /* Limit us to 16 bit value */
    printf ( "0x%04x (Expect 0xFFDC)\n", x );    /* Print in Hex */
}
```

Find size of values in C

exA.c:

```
#include <stdio.h>

int main() {
    unsigned int x1;
    int x2;
    unsigned short x3;
    short x4;
    unsigned long int x5;
    long int x6;

    printf ( "unsigned int   - sizeof %ld\n", sizeof(x1) );
    printf ( "         int   - sizeof %ld\n", sizeof(x2) );
    printf ( "unsigned short - sizeof %ld\n", sizeof(x3) );
    printf ( "         short - sizeof %ld\n", sizeof(x4) );
    printf ( "unsigned long  - sizeof %ld\n", sizeof(x5) );
    printf ( "         long  - sizeof %ld\n", sizeof(x6) );
}
```

Shift and Negative Values

Signed Values

exB.c:

```
#include <stdio.h>

int main() {
    int x;
    x = -10;
    printf ( "0x%08x (Expect 0xFFF*6)\n", x );    /* Print in Hex */
    x >>= 1;
    printf ( "0x%04x (Expect 0xFFF*b)\n", x );    /* Print in Hex */
    printf ( "%d     (Expect -5)\n", x );
}
```

Unsigned makes `>>` a logical shit instead of an arithmetic shift.

exC.c:

```
#include <stdio.h>

int main() {
    unsigned int x;
    x = 0xfffffff6;
    x >>= 1;
    printf ( "0x%04x (Expect 0x7ffffffb)\n", x );    /* Print in Hex */
    printf ( "%d     (Expect 2147483643)\n", x );    /* Print in Hex */
}
```

Assembler Pseudo-Directives

```
ORG

DEC

HEX

OCT
```

hw2.mas as HEX directives:

```
    HEX d014
    HEX 6000
    HEX 1014
    HEX 3015
    HEX 2014
    HEX d014
    HEX 8800
    HEX 9000
    HEX 7000
    HEX 0000
    HEX 0000
    HEX 0000
    HEX 0000
    HEX 0000
    HEX 0000
    HEX 0000
    HEX 0000
    HEX 0000
```

```
      HEX 0000
      HEX 0000
      HEX 0016
      HEX 0001
      HEX 0048
      HEX 0049
      HEX 0050
      HEX 0051
      HEX 5ab2
      HEX aaaa
      HEX 0054
      HEX 0055
      HEX 0056
      HEX 0000
```

or in a more human readable format:

```
  L1,     LoadI    X
          Output
          Load     X
          Add      _1
          Store    X
          LoadI    X
          SkipGt0           / OnLine use Skipins 0x400 - same instruction just different
          Jump L1
          Halt
          ORG  20
  X,      DEC  22            / Counter of how many characters to output.
  _1,     DEC  1
  hw,     HEX  48           / 'H' Your values (clue: 48 is not correct for homework-02!)
      HEX 49                  / 'I'
      HEX 50                  / 'P'
      HEX 51
      HEX 5aB2
      HEX aaaa
      HEX 54
      HEX 55
      HEX 56
      HEX 0
```

Computed GoTo

Reading: https://eli.thegreenplace.net/2012/07/12/computed-goto-for-efficient-dispatch-tables.

exD.c:

```
#include <stdio.h>

#define OP_LOAD     0x1
#define OP_ADD      0x3
#define OP_HALT     0x7

void runVM(unsigned int* mem) {
    unsigned int PC = 0;
    unsigned int IR = 0;
    unsigned int AC = 0;
    unsigned int op, hand;

    while (1) {        // Loop Forever
        IR = mem[PC++];
        op = ( IR & 0xF000 ) >> 12;
        hand = IR & 0x0FFF;
        switch (IR) {
            case OP_LOAD:
                AC = mem[hand];
            break;
            case OP_ADD:
                AC = AC + mem[hand];
            break;
            case OP_HALT:
                return;
            default:
                printf ( "oops - an error 0x%04x not valid\n", IR << 12);
        }
    }
}

void runVM_ComputedGoTo(unsigned int* mem) {
    /* The indices of labels in the dispatch_table are the relevant opcodes */
    static void* dispatch_table[] = {
        &&do_other,       /* 0 */
        &&do_load,        /* 1 */
        &&do_other,       /* 2 */
        &&do_add,         /* 3 */
        &&do_other,       /* 4 */
        &&do_other,       /* 5 */
        &&do_other,       /* 6 */
        &&do_halt,        /* 7 */
        &&do_other,       /* 8 */
        &&do_other,       /* 9 */
        &&do_other,       /* A */
        &&do_other,       /* B */
        &&do_other,       /* C */
        &&do_other,       /* D */
```

```
            &&do_other,       /* E */
            &&do_other,       /* F */
        };
        unsigned int PC = 0;
        unsigned int IR = 0;
        unsigned int AC = 0;
        unsigned int op, hand;


    top:
        IR = mem[PC++];
        op = ( IR & 0xF000 ) >> 12;
        hand = IR & 0x0FFF;
        goto *dispatch_table[op];

        do_halt:
            return;
        do_add:
            AC = AC + mem[hand];
            goto top;
        do_load:
            AC = mem[hand];
            goto top;
        do_other:
            printf ( "oops - an error 0x%04x not valid\n", IR << 12);
            goto top;
    }

    int main() {
        // Placeholder for the moment
    }
```

Copyright