Prof. C’s definition of a match

Definition
The given long sequence $L$ contains the pattern $P$ if and only if after you rotate $L$ left a number of times $r$, $0 \leq r \leq 15$, and then shift it right a number of times $s$, $0 \leq s \leq 15$, you get a result that equals the pattern $P$.

Project 4 Assignment
- Given one pattern $P$ and a zero-terminated sequence of “long” strings, for each “long” string $L$, count the number of times $P$ occurs in $L$ (according the the above definition).
- Note that that count is the number of $r, s$ combinations, $0 \leq r \leq 15, 0 \leq s \leq 15$, for which the above result equals $P$.
- Note details: The counts should be stored after each LC-3 word containing a $P$. Such words have been initialized to 0.
How To Understand A Definition Like This

Definition
The given long sequence $L$ contains the pattern $P$ if and only if after you rotate $L$ left a number of times $r$, $0 \leq r \leq 15$, and then shift it right a number of times $s$, $0 \leq s \leq 15$, you get a result that equals the pattern $P$.

What to DO:

- Make sketches (on paper, etc.) several EXAMPLES.
- The next 2 examples are “generic.”
"Long" string $L$

Pattern $P$

$L_1 = L \cdot \text{LROT. } r$  
($r = 4$)

$L_2 = L_1 \gg s$ ($= 11$)

One match with $r=4$ and $s=11$
Example 2 with Wraparound

"Long" string $L$

Pattern $P$

$L_1 = L \cdot \text{LROT. } r$

($r = 14$)

$L_2 = L_1 >> s$ ($=11$)

One match with $r=14$ and $s=11$
Translation of Assignment into an Algorithm

For each non-zero pattern P, at address A
  For r = 0 to 15
    L1 = L rotated left r bits;
  For s = 0 to 15
    L2 = L1 >> s;  (no sign extension)
    if( L2 == P )
  Halt (trap instruction)
Part of Proj 4 done in C

```c
#include <stdio.h>
unsigned short int rotLeft(unsigned short int x, int n)
{
    while( n-- )
    {
        int sign = x & 0x8000 ;
        x = x << 1;
        if( sign ) { x = x + 1; }
    }
    return x;
}
```
Main function–variables and input

```c
int main(char *argc, char *argv[]) {
    unsigned short int L;
    unsigned short int P[2];

    printf("Long string (Hex): ");
    scanf("%x", &L);
    printf("Pattern (Hex): ");
    scanf("%x", &P[0]);

    P[1] = 0;
    {
        int r;
        int s;
        unsigned short int L1, L2;
    }
}
```
The counting loops

```c
for( r = 0; r <= 15; r++)
{
    L1 = rotLeft(L, r);
    for( s = 0; s <= 15; s++ )
    {
        L2 = L1 >> s;
        if( L2 == P[0] )
        {
            printf(
                "Match: rotations=%d shifts=%d\n", r, s);
            P[1]++;
        }
    }
}
*/end of for( r loop*/

printf(
    "Long string=%#06x Pattern=%#06x Matches=%d\n", 
    L, P[0], P[1]);
} return 0; }
```
Machine Language Programming Tasks

1. Enable a function called with JSR (which uses R7) to call another function with JSR.
   1.1 Plan a memory word for saving and restoring R7’s value.
       (Learn how to use data from memory.)
   1.2 Plan where the ST and LD for saving and restoring R7 will go.
   1.3 Calculate the right PCoffset9 values and code them into the ST and LD instructions.

2. Plan which memory words will be used for the address of the current $P$, and variables such as $r$, $s$, $L1$, $L2$, etc. OR PLAN to use registers for some of them!

3. Learn to program loops. Then plan the 3 nested loops.

4. The top level loop processes a memory array. Learn and code:
   4.1 Access the current $P$ pointed to by its address.
   4.2 Test if $P==0$ and branch out of loop if so.
   4.3 When will the address of the next $P$ be calculated?

5. Initialize $r$ for the next loop. Design and code the loop control.

6. Ditto for the $s$ loop.
ST and LD used to save/restore R7, and other tasks

Illustrated on the blackboard.

▶ You can plan the memory word to save R7 just before the first word of the function. If Fun1 begins at 0x30A0, that word for R7 has address 0x309F.

▶ If you do that, the PCoffset9 is -2.

▶ Code the ST instruction with ?????????? (9?) first. After you figure out its address, then (1) subtract 0x309F-(address of word after the ST instruction.

▶ Code the return with RET.

▶ Some strategies for binary and hex subtraction illustrated.

▶ You can use “scientific mode” of Microsoft’s calculator for Hex arithmetic.
<table>
<thead>
<tr>
<th>Instruction</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD*</td>
<td>0001</td>
<td>DR</td>
<td>SR1</td>
<td>0</td>
<td>00</td>
<td>SR2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD*</td>
<td>0001</td>
<td>DR</td>
<td>SR1</td>
<td>1</td>
<td>imm5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND*</td>
<td>0101</td>
<td>DR</td>
<td>SR1</td>
<td>0</td>
<td>00</td>
<td>SR2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND*</td>
<td>0101</td>
<td>DR</td>
<td>SR1</td>
<td>1</td>
<td>imm5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BR</td>
<td>0000</td>
<td>n</td>
<td>z</td>
<td>p</td>
<td>PCoffset9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP</td>
<td>1100</td>
<td>000</td>
<td>BaseR</td>
<td>000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSR</td>
<td>0100</td>
<td>1</td>
<td>PCoffset11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSRR</td>
<td>0100</td>
<td>0</td>
<td>00</td>
<td>BaseR</td>
<td>000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD*</td>
<td>0010</td>
<td>DR</td>
<td>PCoffset9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDI*</td>
<td>1010</td>
<td>DR</td>
<td>PCoffset9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDR*</td>
<td>0110</td>
<td>DR</td>
<td>BaseR</td>
<td>offset6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEA*</td>
<td>1110</td>
<td>DR</td>
<td>PCoffset9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOT*</td>
<td>1001</td>
<td>DR</td>
<td>SR</td>
<td>111111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RET</td>
<td>1100</td>
<td>000</td>
<td>111</td>
<td>000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTI</td>
<td>1000</td>
<td>000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td>0011</td>
<td>SR</td>
<td>PCoffset9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STI</td>
<td>1011</td>
<td>SR</td>
<td>PCoffset9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STR</td>
<td>0111</td>
<td>SR</td>
<td>BaseR</td>
<td>offset6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRAP</td>
<td>1111</td>
<td>0000</td>
<td>trapvect8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reserved</td>
<td>1101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LC-3 Datapath
Accessing Memory with PC-RELATIVE Addressing

IR: 0010 010 110101111
LD R2 x1AF
PC: 0100 0000 0001 1001
SEXT

MAR: 11111111110101111
MDR: 0000000000000101

ADD

MEMORY

MAR

MDR

R0
R1
R2
R3
R4
R5
R6
R7
R1 should = 001 and R2 = 010. Good for accessing M[A+1] when A is in a register.
Accessing Memory with INDIRECT Addressing

IR: 1010 011 111001100
    LDI  R3  x1CC

PC: 0100 1010 0001 1100

MAR: x2110

MDR:

IR[8:0]: 1010 011 111001100

SEXT:

 ADD:

MEMORY:

R0
R1
R2
R3
R4
R5
R6
R7
A Handy ADDER.. the LEA instruction
LC-3 Branch Control Circuitry

PC: 0100 0000 0010 1000
IR: 0000 011011001
N Z P PCoffset9 BR
0000 011011001 9
Yes!
PZN
0 1 0
PC 0100 0000 0010 1000
0100 0001 0000 0001

SEXT
16
16
0000000011011001
ADD
0100 0000 0010 1000
0100 0001 0000 0001
PCMUX
Counter Controlled Array Processing Loop

R1 ← x3100
R3 ← 0
R2 ← 12

Yes R2 ? = 0
No

R4 ← M[R1]
R3 ← R3 + R4
Increment R1
Decrement R2
Sentinal Controlled Array Processing Loop

R1 ← x3100
R3 ← 0
R4 ← M[R1]

Yes

R4 ?= Sentinel

No

R3 ← R3 + R4
Increment R1
R4 ← M[R1]

R4 ← M[R1]