Organização de Computadores Digitais - Aula de 8-8-17

Como é feita a transição do código em C para o Assembly?

C

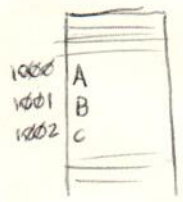
```
unsigned char a=5;
unsigned char b='G';
unsigned char c;
c = a+b;
if (c>10){
    c=10;
}
```

comp.

Assembly

```
1  LOAD R0 #5 *
2  STORE 1000 R0
3  LOAD R0 #G
4  STORE 1001 R0
5  LOAD R0 1000
6  LOAD R1 1001
7  ADD R0 R0 R1
8  STORE 1002 R0
9  LOAD R0 #10
10 LOAD R1 1002
11 CMP R1 R0 **
12 JEL 14 ***
13 STORE 1002 R0
14
```

Estamos assumindo que usamos um Arduino, já que nele não é o S.O. que cuida da memória.



Isso poderia ser otimizado não usando o R0 todas as vezes. Por exemplo:

```
Load R0 #5
STORE 1000 R0
Load R1 #G
STORE 1001 R1
ADD R2 R0 R1
STORE 1002 R2
```

* O # indica que o que vem a seguir é um número. A ausência desse marcador denota um endereço, o registrador receberá o conteúdo do End.

** A comparação ocorre do primeiro em relação ao segundo. O resultado disso é representado em flags.*

*** O JMP é um goto. Há variações para situações específicas:

- JGR: jump if greater, (checa a flag de maior)
- JLE: jump if lesser,
- JEQ: jump if equal;
- JEG: jump if equal or greater;
- JEL: jump if equal or lesser...

O FR* a grosso modo é um conjunto de flip-flop que guarda informações específicas. Algumas flags existentes: flag de carry, de maior, de menor, de igual, de 0, de divisão por 0, de overflow, etc. → Flags

EXEMPLO: Otimização

C

```
for(i=0; i<10.000; i++)
```

Assembly

```
LOAD R0 #1
LOAD R1 #0
LOAD R2 #10000
X: ADD R1 R1 R0 *
CMP R1 R2
JEL X
```

Otimizando

```
LOAD R1 #10.000
LOAD R0 #1
X:
SUB R1 R1 R0 **
JNZ X ***
```

* i++

** Quando a operação der 0, sobe a flag de 0.

*** jump if not zero