

- simple if (assume first instruction is at address 20)

```

load    #0      ;   if (0 > x) {
cmp     x       ;
ble     endif1  ;
load    #0      ;   x = 0 - x;
sub     x       ;
store   x       ;
endif1: ;   }

```

- what happens to the PSW (for Memory[x]=3)?

```

load #0      → GT=0, EQ=0, PC=21
cmp x       → GT=0, EQ=0, PC=22
ble endif1  → GT=0, EQ=0, PC=26 (sets PC to endif1 if GT=0)

```

- for simple C, we have standard translation patterns
- expression evaluation similar like assignment, except use `cmp` instead of `add`, `sub` etc
- use the *opposite* branch instruction to the condition
- use systematically-named branch targets
- for `if`, we need to be able to do a conditional forward branch, e.g. `PC = endif`, if `GT=0`
- for `else if/else`, we also need an unconditional forward branch (`jmp`)
- the PSW plays an important role in all control structures

Translating C into PeANUt – if .. else if .. else ...

```

load    sal     ;   if (sal <= 250) {
cmp     #250   ;
bgt     elsif2  ;
load    #0     ;   tax = 0;
store   tax    ;
jmp     endif2  ; }
elsif2: load    sal     ;   else if (sal <= 500) {
cmp     #500   ;
bgt     else2   ;
load    sal     ;   tax = sal*20;
mul     #20    ;
store   tax    ;
jmp     endif2  ; }
else2:  ;   else {
load    sal     ;   tax = sal*30;
mul     #30    ;
store   tax    ;
endif2: ...    ;   }

```

Translation of Simple I/O into PeANUt

- Scan and print characters

```

ch:   block 1      ;   char ch; /* 16 bits! */
      ;
      trap   #2    ;   ch = getchar();
      store  ch    ;
      ;
      load   ch    ;   printf("%c", ch);
      trap  #3    ;   // same as putchar(ch)

```

- basic I/O is via traps, i.e. the operating system
- in PeANUt, the value `-1 (0xffff)` gets stored in AC to signify end-of-file (as in C)
- always make sure you print ASCII values (not integer numbers)
- question: how do we do more complex I/O in PeANUt?