

Further PeANUt

- Ref: [PeANUt Spec,]; also [O'H&Bryant, sect 3.6]
- traps:
 - input and output
 - relationship with exceptions
- conditional branches:
 - condition codes (part of PSW register)
 - branch instructions
 - repetition
- index register and indexed addressing mode
- other issues:
 - revise 2007 MSE, Q2(a)(ii) (C Programming)
 - origin of 'PeANUt'

Traps

- traps are operations that cannot be performed with the normal PeANUt instructions (e.g. print, read from keyboard, or stop a program)
 - there is a special instruction called **trap**, which is used to have PeANUt perform a service that is usually a call to the operating system
 - its opcode is **110101**, its operand is the trap number
 - depending on the trap number, some particular action will be performed
 - Halt: tells the PeANUt where your program ends
 - Get: allows you to read a character from keyboard
 - Put: allows you to print out a character
- | | action | opcode | operand |
|--|--------|--------|---------|
| | Halt | 110101 | 1 |
| | Get | 110101 | 2 |
| | Put | 110101 | 3 |
- extension of the instruction set

Traps – Input / Output Example: echoline.mli

```
; Simple example machine language program to read and print a line
START a10 ; start address of the program, initialise PC to this

AT a0
0000 0000 0000 0000 ; a0: memory cell to store the character

AT a10 ; store the following data items (instrns)
; into memory from a10 onwards
110101 0000 0000 10 ; a10: trap 2 (Get), read character
001 010 0000 0000 00 ; a11: store a0, store the character
000 111 0000 0010 10 ; a12: compare EOL (End-Of-Line character)
101001 0000 0011 10 ; a13: branch equal a16
110101 0000 0000 11 ; a14: trap 3 (Put), print character
101000 0000 0010 00 ; a15: jump to a10, jump to start of loop
110101 0000 0000 11 ; a16: trap 3 (Put), print EOL character
110101 0000 0000 01 ; a17: trap 1 (Halt), stop the program
```

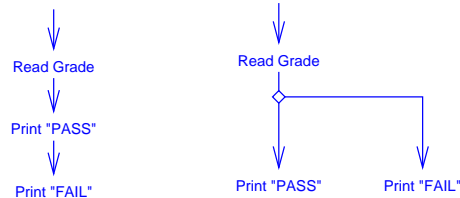
(use this as starting point for assignment 2, part 1)

Traps and Exceptions

- the terms trap and exception are also used to describe the occurrence of a failure condition and PeANUt's response to it (more on traps in lecture P9)
- important PeANUt-initiated exceptions:
 - 5: Illegal Instruction: when it is asked to execute an illegal instruction
 - 6: Illegal Mode: when it is asked to execute an instruction with an invalid mode
 - 7: Overflow: when an arithmetic overflow occurs while the overflow enable bit (EN) bit in CC (condition codes) is set
 - 8: Divide by Zero
- the same effect can be achieved by executing a trap instruction with the corresponding trap number
- Q: why is this an important mechanism to have in a machine?

Conditional Branches

- a conditional branch is a decision point for an instruction sequence
- in C: `if (cond) {...} else {...}`
- are conditional branches necessary?
 - imagine an instruction sequence that, after reading in a grade, outputs either the word "PASS" or the word "FAIL"



- so far, the PeANut has only been allowed to execute a continuous instruction sequence, unchanging over each execution of the program
- the machine must be able to behave differently under different situations

Conditional Branches Example

- we want to write a program that performs:

$\text{mem}[a30] \leftarrow \text{mem}[a20] + \text{mem}[a25]$, but only if $\text{mem}[a20] \leq 17$

```

001 001 0 000 010 000 ; a5: load mem[a20]
000 111 0 000 010 001 ; a6: compare AC with 17 (GT=1 if AC>17)
101011 0 000 001 010 ; a7: branch if greater (to a12 if GT=1)
001 011 0 000 010 101 ; a10: add mem[a25]
001 010 0 000 011 000 ; a11: store mem[a30]
... more code ... ; a12: ...
    
```

- assume $\text{mem}[a25] = 10_{10}$; consider execution for:

- $\text{mem}[a20] = 0$
- $\text{mem}[a20] = 20_{10}$

Condition Codes and Branches

- the condition codes: GT (greater than), EQ (equal) and OV (overflow)
 - bits within the program status word (PSW) (bits 12, 11 and 10)
 - a compare instruction sets the CCs:
 - ◆ GT: set to 1 if $AC > OP$ (operand), otherwise
 - ◆ EQ: set to 1 if $AC = OP$ (operand), otherwise
 - ◆ OV: set to 1 if last ALU operation resulted in an overflow, else 0
- conditional branch instructions:
 - allow a new value for the PC to be set, depending on the GT, OV or EQ bits
 - are normally executed immediately after a compare instruction
 - what happens to the PC?
 - ◆ $PC \leftarrow OP$ (operand), if the branch is taken
 - ◆ PC remains unchanged otherwise (points to the next instruction)

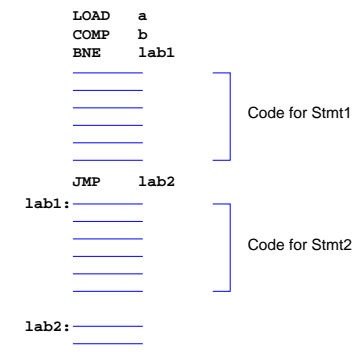
- branches available:

branch	taken when:	branch	taken when:
BranchEqual	EQ = 1	BranchLessEqual	GT = 0
BranchNotEqual	EQ = 0	BranchOverflow	OV = 1
BranchGreater	GT = 1	Jump	ALWAYS

Extension to If – Then – Else

- we should consider the usual conditional:

`if (a == b) { <Stmt1> } else { <Stmt2> }`



- i.e. at the end of each `<Stmt>`, we insert a jump instruction to go to the end of the conditional

