

More PeANUt

- PeANUt initialisation
- Basic programming
 - Addition
 - Logical And
 - Subtraction
- Reference:
 - Specification of the PeANUt computer (Section 2, Appendix C)
- Additional reading: Sections 3.1 to 3.5 in Bryant and O'Hallaron

More on Initialising

- To initialise the PeANUt, we have to define the state of the memory, the PC and set the other registers to zero
- How is this initialisation performed?
 - With a file specifying the desired state
- The *.mli file (**m**achine language **i**nitialisation file)
 - Create a file *myfile.mli*
 - Provide start PC value by **START** <address>
 - Not every memory cell needs to be mentioned
 - Define memory by listing cell contents
 - Each such list has a starting point specified by **AT** <address>

Initialising the PeANUt

- A microprocessor automatically executes a sequence of instructions
- For useful operation, we need to *initialise* a microprocessor, by defining its *starting state*
- Initialisation Process:
 - How is the state of the PeANUt defined?
 1. The state of the memory
 2. The state of the PC (program counter) (*program start*)
 3. The state of other registers

Machine language initialisation file

- **addition.mli**

```
START a10      ; Put address 108 into the PC
AT a10         ; From address 108 on, write . . .
001 001 0 000 000 001 ; a10: load mem[a1]
001 011 0 000 000 010 ; a11: add mem[a2]
001 010 0 000 000 011 ; a12: store mem[a3]
110101 0 000 000 001 ; a13: trap 1 (halt)
```
- How is the PeANUt initialised?
 - Line of the form **START aX**, where X is an octal number. It specifies the initial program counter (PC) value
 - Lines of the form **AT aX**, where X is an octal number. It is the starting address of a block of memory cells into which the following data items go.
 - Data values. A suitable format is 16 bits

More on machine language initialisation files

- Some things are ignored
 - Comments: Anything written to the right of a ;
 - Blank lines
 - Spaces and tabs in data values
- Restrictions
 - There can only be one **START** line
 - There may be many **AT**'s, but,
 - They must not result in overlapping data
 - They must appear in sequence
- *.mli files are converted to *.img (image) files (binary files)
 - Image files can directly initialise your PeANUt
 - *.mli files cannot!

5

COMP2300, 2006

An Example

```
; Simple example machine language program to print a word
; Program: printword.mli

START a10 ; Start address of the program, initialise PC to this

AT a10 ; Store the following data items (instructions) into
; memory from a10 onwards
000 001 0 001 001 000 ; a10: Load 'H' (immediate mode)
110101 0 000 000 011 ; a11: trap 3 (put)
000 001 0 001 000 101 ; a12: Load 'E' (immediate mode)
110101 0 000 000 011 ; a13: trap 3 (put)
000 001 0 001 001 100 ; a14: Load 'L' (immediate mode)
110101 0 000 000 011 ; a15: trap 3 (put)
000 001 0 001 001 100 ; a16: Load 'L' (immediate mode)
110101 0 000 000 011 ; a17: trap 3 (put)
000 001 0 001 001 111 ; a20: Load 'D' (immediate mode)
110101 0 000 000 011 ; a21: trap 3 (put)
000 001 0 000 001 010 ; a22: Load '\n' (new line) (immediate mode)
110101 0 000 000 011 ; a23: trap 3 (put)
110101 0 000 000 001 ; a24: trap 1 (halt)
```

6

COMP2300, 2006

Basic programming

- Arithmetic and logic on an accumulator style architecture
- Instructions are available in different modes:
 - 000 Immediate mode
 - 001 Direct mode
 - 010 Indirect mode
 - 011 Indexed mode (*later*)
 - 100 Stack mode (*later*)
- Each program needs a **halt** instruction at the end to make it complete (trap 1)

7

COMP2300, 2006

Addition (immediate mode)

- $\text{mem}[a35] \leftarrow 5 + 14$

instruction	AC	mem[a35]
load 5	5	?
add 14	19	?
store mem[a35]	19	19
halt		

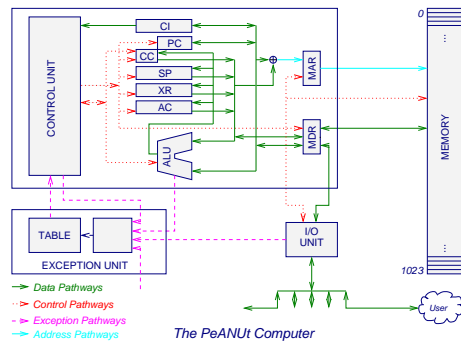
- Machine code (need to include **AT** and **START** to make it valid)


```
000 001 0 000 000 101 ; load 5
000 011 0 000 001 110 ; add 14
001 010 0 000 011 101 ; store mem[a35]
110101 0 000 000 001 ; halt (trap 1)
```

8

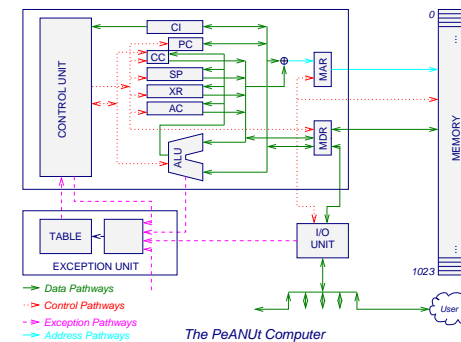
COMP2300, 2006

Addition (immediate mode) – Inside the machine



- **load 5**
AC ← CI[9..0]
- **add 14**
ALU add
AC ← AC + CI[9..0]
- **store mem[a35]**
MAR ← CI[9..0]
MDR ← AC
Write, Enable
mem[a35] ← MDR

Addition (direct mode) – Inside the machine



- **load mem[a1]**
MAR ← CI[9..0]
Read, Enable
MDR ← mem[a1]
AC ← MDR
- **add mem[a2]**
MAR ← CI[9..0]
Read, Enable
MDR ← mem[a2]
ALU add
AC ← AC + MDR
- **store mem[a3]**
MAR ← CI[9..0]
MDR ← AC
Write, Enable
mem[a3] ← MDR

Addition (direct mode)

- mem[a3] ← mem[a1] + mem[a2]
Say mem[a1] = 4, mem[a2] = 5

instruction	AC	mem[a3]
load mem[a1]	4	?
add mem[a2]	9	?
store mem[a3]	9	9

- Machine code (need to include **AT** and **START**)
001 001 0 000 000 001 ; load mem[a1]
001 011 0 000 000 010 ; add mem[a2]
001 010 0 000 000 011 ; store mem[a3]
110101 0 000 000 001 ; halt (trap 1)

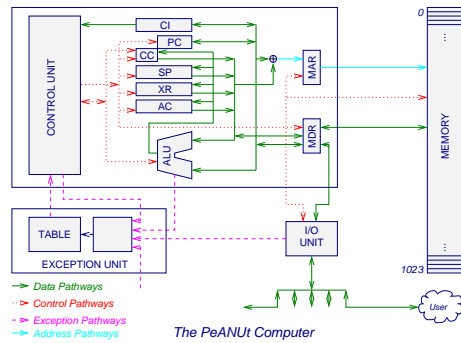
Logical And (direct mode)

- mem[a1] ← mem[a63] AND mem[a17]
mem[a63] = 000 000 00000 11100
mem[a17] = 101 010 10101 01010

instruction	AC	mem[a1]
load mem[a63]	..011100	?
and mem[a17]	..001000	?
store mem[a1]	..001000	..001000

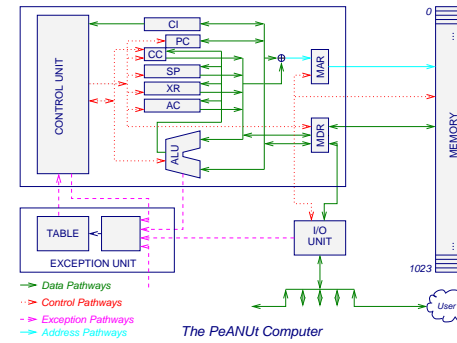
- Machine code (need to include **AT** and **START** to make it valid)
001 001 0 000 110 011 ; load mem[a63]
101110 0 000 001 111 ; and mem[a17]
001 010 0 000 000 001 ; store mem[a1]
110101 0 000 000 001 ; halt (trap 1)

Logical And (direct mode) – Inside the machine



- **load mem[a63]**
 $MAR \leftarrow CI[9..0]$
 Read, Enable
 $MDR \leftarrow mem[a63]$
 $AC \leftarrow MDR$
- **and mem[a17]**
 $MAR \leftarrow CI[9..0]$
 Read, Enable
 $MDR \leftarrow mem[a17]$
 ALU and
 $AC \leftarrow AC \text{ AND } MDR$
- **store mem[a1]**
 $MAR \leftarrow CI[9..0]$
 $MDR \leftarrow AC$
 Write, Enable
 $mem[a1] \leftarrow MDR$

Subtraction (indirect mode) – Inside the machine



- **load 20**
 $AC \leftarrow CI[9..0]$
- **sub mem[mem[a10]]**
 $MAR \leftarrow CI[9..0]$
 Read, Enable
 $MDR \leftarrow mem[a10]$
 $MAR \leftarrow MDR[9..0]$
 Read, Enable
 $MDR \leftarrow mem[mem[a10]]$ (mem[a52])
 ALU sub
 $AC \leftarrow AC - MDR$
- **store mem[a23]**
 $MAR \leftarrow CI[9..0]$
 $MDR \leftarrow AC$
 Write, Enable
 $mem[a23] \leftarrow MDR$

Subtraction (indirect mode)

- $mem[a23] \leftarrow 20 - mem[mem[a10]]$
 $mem[a10] = 000\ 000\ 0\ 000\ 101\ 010$ (a52)
 $mem[a52] = 000\ 000\ 0\ 000\ 011\ 110$

instruction	AC	mem[a23]
load 20	20	?
sub mem[mem[a10]]	-10	?
store mem[a23]	-10	-10

- Machine code (need to include **AT** and **START** to make it valid)
 $000\ 001\ 0\ 000\ 010\ 100$; load 20
 $010\ 100\ 0\ 000\ 001\ 000$; sub mem[mem[a10]] (mem[a52])
 $001\ 010\ 0\ 000\ 010\ 011$; store mem[a23]
 $110101\ 0\ 000\ 000\ 001$; halt (trap 1)