## Task Consists Of

* Simulator
  + load machine code program from file
  + demonstrate step-by-step execution
  + illustrate instruction decode + operand
  + show the state of the registers and memory after each cycle
* Assembler
  + reads in an assembly language (text) file and converts to machine code
  + outputs the machine code as a file which can be run by your simulator
  + your demonstration should include editing the source file then re-assembling to show changes in the program

## Methods

* 2 Programs : Assembler | Simulator
* Assembler
  + Read in machine code [as bit set?]
  + Translate to binary
  + Output to binary file
  + Call Simulator e.g. ./BabySim.cpp

* Simulator
  + Main:
  + increment\_CI( )
  + fetch( )
  + decode( )
  + execute( )
  + display\_everything( )
* Classes

Store [32 lines, each line\* holds 32 bits / bits numbered 0-31 from left to right]

* + Read in source code
  + & store in store bitset[fetch]
  + Dump Store to Screen

Accumulator [single line (register) of 32 bits]

* + Arithmetic
  + Instruction Set

Control [CI Control Instruction |PI Present Instruction]

* + Initiate int CI = 0; //line 1 first executed | bitset problem? CI needs to be binary?
  + CIAccessor //return CI line number of store next instruction
  + CIMutator : Increase bitset CI by 1
  + PIAccessor //return PI instruction executed this cycle
  + PIMutator //store next line from store in PI
  + decode( )
  + execute( )
  + display\_everything( )

## We Need to Implement



## Task Allocation

* Andrew –
  + Assembler
* Jüri –
  + Bitset
  + FileHandling
  + Store
* Peter S –
  + Bin to Dec
  + Dec to Bin
* Peter W –
  + CI
  + Code Compiler

Final Functionality

## Tests to Carry Out & Aim For



## Pseudocode

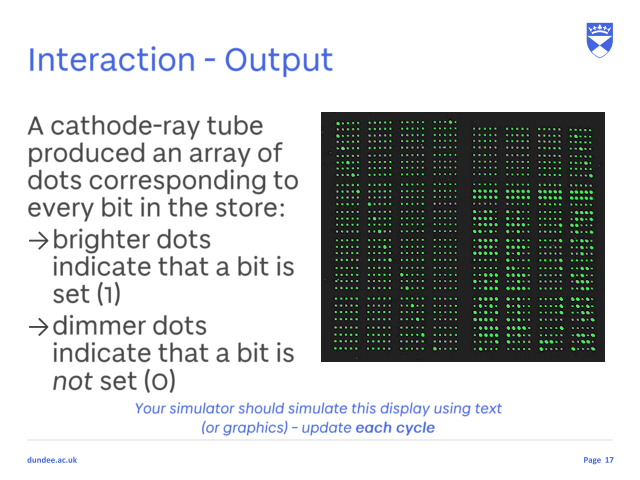
* + increment\_CI( )
  + fetch( )
  + decode( )
  + execute( )
  + display\_everything( )
  + UNTIL halted

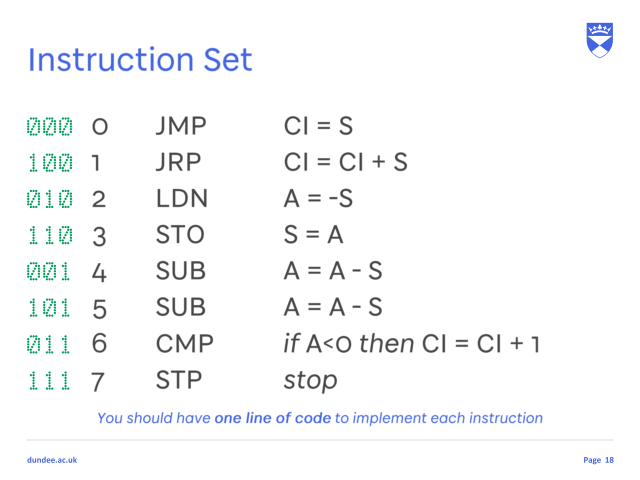
# Notes

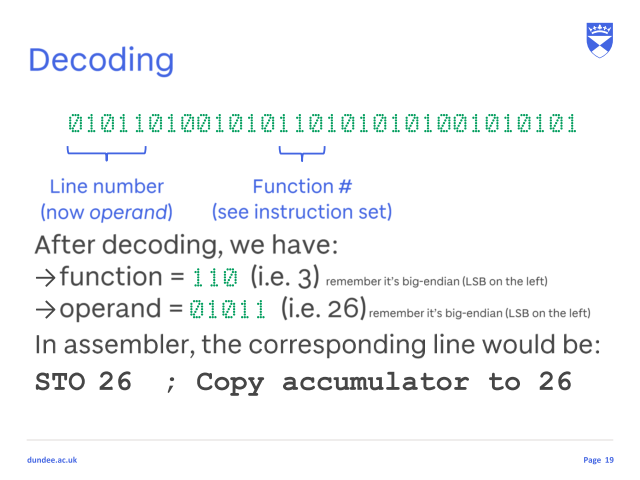
* executed the world’s first electronically-stored computer program
* the **Store**
* 32 lines, each line\* holds 32 bits
* bits numbered 0-31 from left to right\*\*
* the **Accumulator**
* **single** line (**register**) of 32 bits
* holds results of arithmetic operations
* the **Control**

\* what we now call a memory location \*\* big-endian

* 40 buttons to manually write data into the store:
* ⦁ originally intended to correspond to 40 bits of memory (but only 32 used)
* ⦁ directly manipulated contents of the store
* ⦁ **Your simulator will load the whole machine code program directly from a text file on disk!**

* **Your simulator should simulate this display using text (or graphics) – update each cycle**
* 

 **CI, S, A**

* **Function [13-15] | Operand [1-5]**
* **This was the initial 32x32 store content loaded from paper tape (simulated by your hard disk).**

Processor constantly does fetch-execute cycle, so your main program could be just:

{

REPEAT

increment\_CI( )

fetch( )

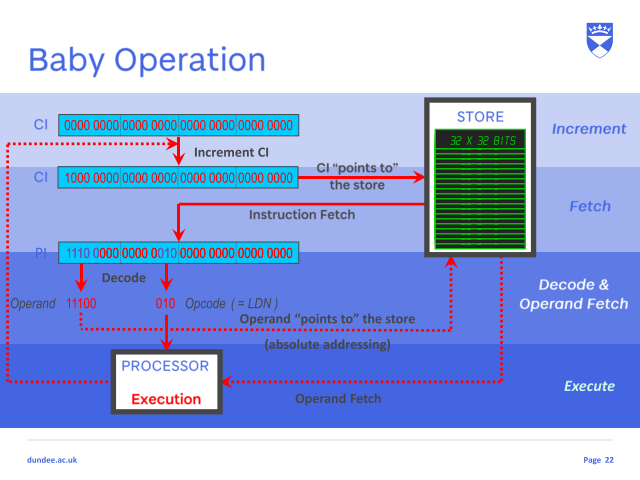
decode( )

execute( )

display\_everything( )

UNTIL halted

}

* Operation
* **Task**

Research the history and structure of the Manchester Baby (key documents are on MyDundee) to develop a full understanding of the fetch-execute cycle

Using C or C++ in Linux, develop a program which simulates the operation of the Baby:

load machine code program from file

demonstrate step-by-step execution

illustrate instruction decode + operand

show the state of the registers and memory after each cycle

Using C or C++ in Linux, develop an assembler program for the Baby:

reads in an assembly language (text) file and converts to machine code

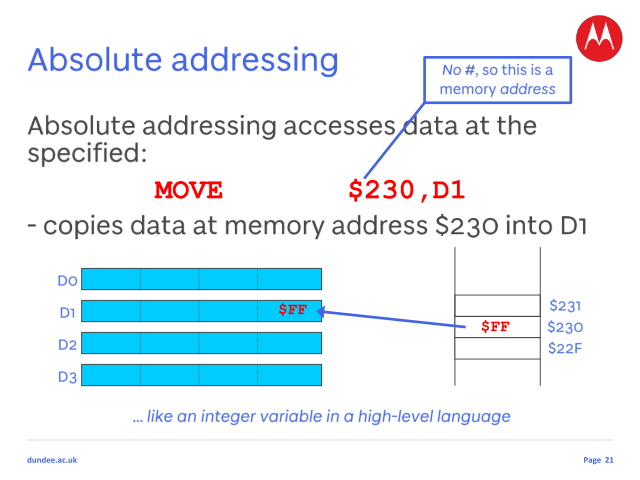
outputs the machine code as a file which can be run by your simulator

your demonstration should include editing the source file then re-assembling to show changes in the program

* **Program**

2 Programs : Assembler | Simulator

Both use machine code file so need to decide file format for this file  
Assembler program is easier?

2 Complament

Int in

Out : 1+decToBin((2^31-[1:endin]))

Minus : 1

For (i=0; i=sizeOut-1; i++){

Minus

}

//binary to decimal string

int Operation::binToDecimal(string binary)

{

if (binary[0] == 1){

string temp = binary[1-end]

//The number starts with 1, so it's negative, so we find the complement of 1111, which is 0000. invert 0's and 1's

For (i=0; i<temp.getSize(); i++){

If tmp[i] == ‘1’ then

tmp[i] = 0;

else

tmp[i] = 1;

}

//Add 1 to 0000, and we obtain 0001.

// 0011000111 we need to flip all the bits after the rightmost 0 bit (we get 001100**0**000). Finally, flip the rightmost 0 bit also (we get 0011001000) and then have added 1

 //flip

int location = -1;

for (i=temp.getSize(); i>-1; i--){

if (tmp[i] == “0”) then{

location = i;

break;

}

}

if (i=-1){

do something???

}else{

for (o=location; i<temp.getSize()){

if (tmp[i] == “0”) then{

tmp[i] = 1;

}else{

tmp[i] = 0;

}

}

}

//covert tmp to decimal

string binary = tmp

//Convert 0001 to decimal, which is 1.

int value = atoi(binary.c\_str()); // converst string into integer.

//Apply the sign = -1.

return Operation::binToDecimal(-value);

}

else{

int value = atoi(binary.c\_str()); // converst string into integer.

return Operation::binToDecimal(value);

}

}

//binary to decimal int

int Operation::binToDecimal(int binary)

{

int value = binary;

int decimalNumber = 0, i = 0, remainder;

while (value != 0)

{

remainder = value % 10;

value /= 10;

decimalNumber += remainder\*pow(2, i);

++i;

}

return decimalNumber;

}

//decimal string to binary

string Operation::decToBin(string dec)

{

// Convert the number to binary (ignore the sign for now) e.g. 5 is 0101 and -5 is 0101

If the number is a positive number then you are done. e.g. 5 is 0101 in binary using twos complement notation.

If the number is negative then

3.1 find the complement (invert 0's and 1's) e.g. -5 is 0101 so finding the complement is 1010

3.2 Add 1 to the complement 1010 + 1 = 1011. Therefore, -5 in two's complement is 1011.

int value = atoi(dec.c\_str()); // converst string into integer.

return Operation::decToBin(value);

}

//binary to decimal int

string Operation::decToBin(int dec)

{

string binary;

int mask = 1;

if (dec < 0){

//2^31+(-23)

int out = pow(2, 31);

std::cout << "2^31 : pow(2, 31)" << out << std::endl;

binary = "1" + decToBin(pow(2, 31)+dec);

}else{

for (int i = 0; i < 32; i++)

{

if ((mask&dec) >= 1)

binary = "1" + binary;

else

binary = "0" + binary;

mask <<= 1;

}

}

return binary;

}