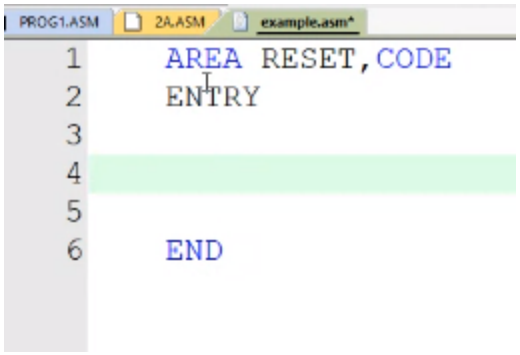


ARM HAS A BARREL SHIFTER, SUPPORTS LEFT AND RIGHT SHIFT

FASTEST WAY TO MULTIPLY BY 2, IT IS BY LEFT SHIFT

ASSEMBLY IS A LOW-LEVEL LANGUAGE

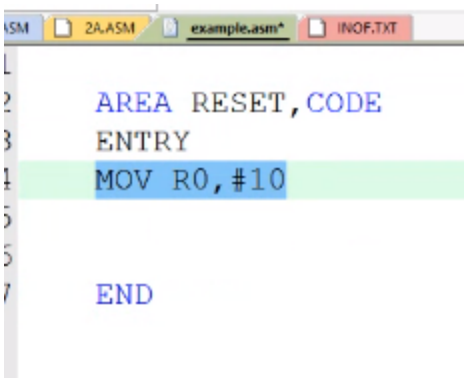
THIS HIGH LEVEL OR LOW LEVEL MUST BE CONVERTED TO BINARY, THE BINARY CODE IS FETCHED AND EXECUTED

Instruction to the assembler (above is an instruction to assembler only)

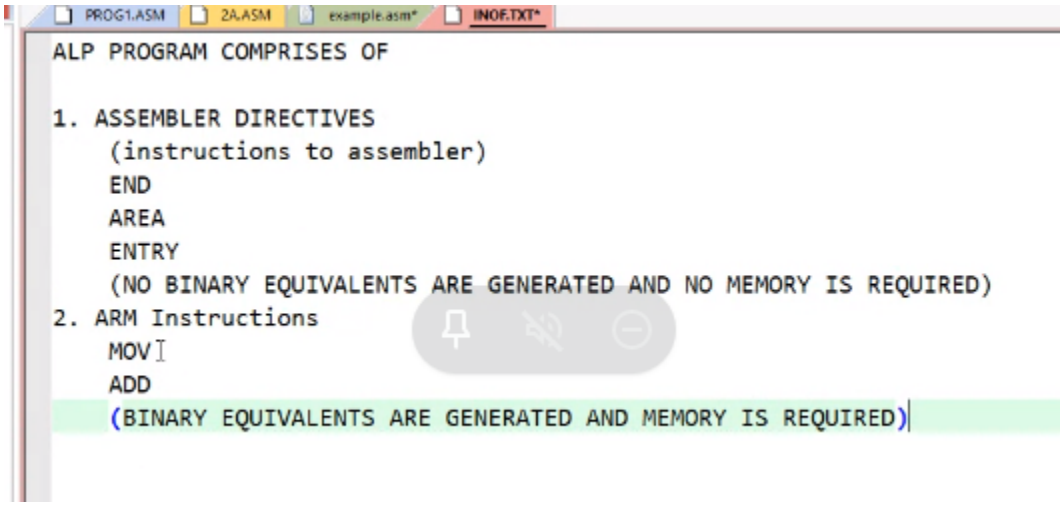
- They are called **assembler directives**
- They do not have binary equivalents, i.e not converted to binary
- No memory is NEEDED
- Simply to be followed by the assembler
- Helps convert the code in assembly to the binary code
- Is not a part of the code, just aids in the process
- Has significance during the time of writing the program and conversion

Instruction to arm core

- Have an equivalent binary code represented in the memory
- Hex code will be generated
- Need memory
-



This has 3 instructions to the assembler - AREA RESET, CODE, ENTRY, END
One to the arm core - MOV, ADD



THE HEX EQUIVALENT IS USED INSTEAD OF ALL THE 1S AND 0S

THIS IS WHAT IS ACTUALLY PRESENT
0000 0000 0000 0000 0000 0000 0000 0000

EACH BLOCK CAN TAKE VALUES FROM 0 TO FOR 0 TO 15

0 0000

1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

DISASSEMBLY

IDENTIFYING THE HEX/BINARY CODE OF EVERY INSTRUCTION
4 MEMORY LOCATIONS TO REPRESENT 1 INSTRUCTION -> AS EACH IS 32 BITS
EACH BYTE HAS 8 BITS

E3A0 000A

1110 0011 1010 0000 0000 0000 0000 1010

E 3 A 0 0 0 0 A

ASSEMBLER TAKES EVERY INSTRUCTION AND MAKES IT INTO THIS FORMAT
EVERY INSTRUCTION IS CONVERTED TO A SINGLE BINARY PATTERN

ONE TO ONE CORRESPONDENCE TO THE INSTRUCTION
AS EACH INSTRUCTION IS BROKEN DOWN AND REPRESENTED

HOWEVER IN A HIGH-LEVEL LANGUAGE,

Int a = b + c;
Gives a set of add and move instructions

HIGH-LEVEL -> ASSEMBLY INSTRUCTIONS -> MACHINE LANGUAGE

So no one to one correspondence exists in a high-level language

```
what is the machine language equvivalent of MOV R0,#10
E3A0000A
11100011101000000000000000001010

0 - 0x0000 0000
1 - 0x0000 0001
2 - 0x0000 0010
3 - 0x0000 0011
```

These 32 bits or 4 bytes is used to represent the instruction

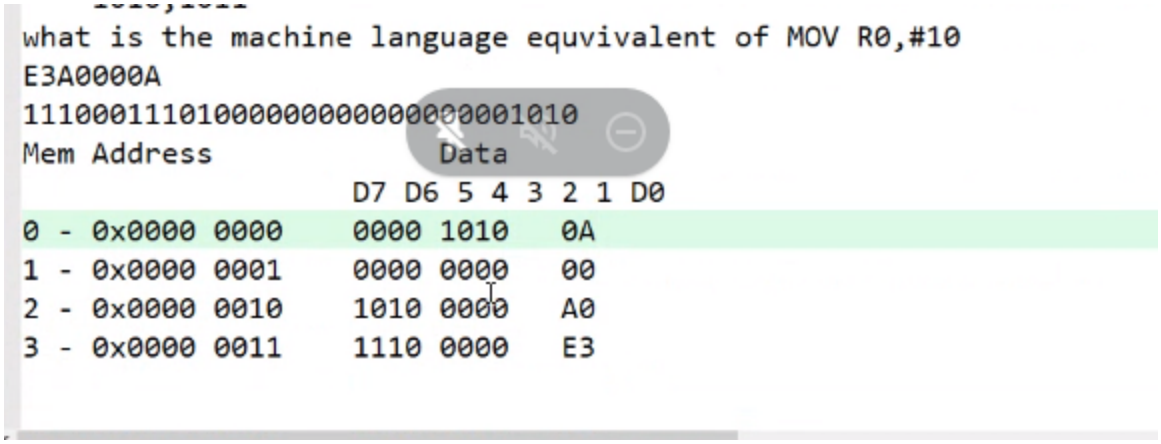
```
0 - 0x0000 0000
1 - 0x0000 0001
2 - 0x0000 0010
3 - 0x0000 0011
```

Loc 3 - >	1110	0011	EA
Loc 2 - >	1010	0000	A0
Loc 1 ->	0000	0000	00
Loc 0 - >	0000	1010	0A

REPRESENTED AS

IN THE ACTUAL MEMORY
0A 00 A0 EA

This follows **little-endian**, as the LSB bits come before the MSB bits



MEMORY

TOTAL 2**32 -> 4 GB , FOR ALL PURPOSES

THOUGH THERE IS SUPPORT FOR 4GB ONLY 512 KB OF FLASH, 32 KB OF SRAM

FLASH IS PUT FOR THE FIRST 512 KB, AS OUT ADDRESSES START FROM

0X00000000 TO 0X8000

IN LPC 2148

512 KB -> FLASH

32 KB -> RAM

FROM 0 ONWARDS, THE FLASH STARTS AND HAS A SIZE OF 512 KILO BYTES

FROM 1 GB ONWARD THE SRAM STARTS HAS A SIZE OF 32 KILOBYTES

INSTRUCTIONS

- Types

Data manipulation - 3 ADDRESS FORMAT

Data transfer
Arithmetic operations
Logical
Shift
rotate

Branching

B (stop B stop)

Load and store

These are the only instruction can access memory or SRAM

ONLY LOAD AND STORE USES OR CAN ACCESS MAIN MEMORY, THE REST CAN ACCESS ONLY REGISTERS, THIS IS WHY IT IS CALLED LOAD AND STORE ARCHITECTURE

VARIABLES

- STORE VARIABLES EITHER IN REGISTERS

- STORE IN SRAM

FIRSTLY TRY TO LOOK AT WHETHER UR REGISTER IS USABLE