Assembly language Presentation

3rd year electronics and communication

Names of our team:

1-Ibrahim Mohamed Salah

2-Ahmed Hesham Zaki

3-Ali Ayman Sakr

4-Ahmed Elhosiny Sobhy

5-Ahmed Gamal

**Advantages Assembly Language**

1.The symbolic programming of **Assembly Language** is easier to understand and saves a lot of time and effort of the programmer.

2.It is easier to correct errors and modify program instructions.

3.**Assembly Language** has the same efficiency of execution as the machine level **language**.

Disadvantages of assembly language

1-The assembler does not solve all the problems of programming. (One problem is the tremendous gap between the microcomputer instruction set and the tasks which the microcomputer is to perform. Computer instructions tend to do things like add the contents of two registers, shift the contents of the accumulator one bit, or place a new value in the progam counter. On the other hand, a user generally wants a microcomputer to do something like print a number, look for and react to a particular command from a teleypewriter, or activate a relay at ther proper time).

2-An assembly language programmer must translate such tasks into sequence of simple computer instructions. The translation can be adifficult, time-consuming job.

3- if you are programming in assembly language, you must have detailed knowledge of the particular microcomputer that you are using.

4-You must know what registers and instructions the microcomputers has, precisely how the instructions affect the various registers, what addressing methods the computer uses, and a mass of other information. None of this information id relevant to the task which the microcomputer must ultimately perform.

5-assembly language programs are not portable. Each microcomputer has its own assembly language which reflects its own architecture. And assembly language program written for arm will not run on a 486, pentium, or z8000 microprocessor.

ARMulator Assembler

-This is necessary elements of an assembler program for the armulator, notice that this, and all other example programs have the following elements. Firstly there must be an ENTRYdirective. This tells the assembler where the first executable instruction is located. Next there must be at least one AREA directive, at the start of the program, and there may be other area directives to define data storage areas. Finally there must be an END directive, to show where the code ends. The absence of any of these will cause the assembly to fail with an error.

First program

Find the larger of two numbers

TTL comparenum (name of your program)

AREA program(any name), CODE, READONLY

ENTRY

Main

LDR R1, Value1 ;load the first value to be compared

LDR R2, Value2 ;load the second value to be compared

CMP R1, R2 ;Compare them

BHI Done ;if R1 contains the highest

MOV R1, R2 ;otherwise overwrite R1

Done

STR R1, Result ;store the result

SWI &11 ;exit

Value1 DCD &FEDCA987 ;Value to be compared

Value2 DCD &12345678 ;Value to be compared

Result DCD 0 ;Space to store result

END

Explantion of code

The compare instruction, CMP, sets the status register flags as if the destination, R1, were subtracted from the source R2.

In AREA:

CODE : Contains machine instructions. READONLY is the default.

READONLY :Indicates that this area should not be written to.

Compare condition signed undigned

Greater than BGT BHI

 DCD: directive allocates one or more words(=4bytes) of memory, aligned on four-byte boundaries, and defines the initial runtime contents of the memory

Second program factorial

TTL factorial

AREA program,CODE, READONLY

ENTRY

Main

LDR R0, =DataTable ;load the address of the lookup table

LDR R1, Value ;offest of value to be looked up

MOV R1,R1,LSL#0x2 ;data is declared as 32bit – need

;to quadruple the offest to point at the ;correct memory location

ADD R0, R0, R1 ;R0 now contains memory address to store

LDR R2, [R0]

ADR R3, Result ;the address where we want to store the answer

STR R2,[R3] ;store the answer

SWI &11

AREA DataTable, DATA

DCD 1 ;0! = 1 ;the data table containing the factorials

DCD 1 ;1! =1

DCD 2 ;2!=2

DCD 6 ;3!=6

DCD 24 ;4! = 24

DCD 120 ;5! = 120

DCD 720 ;6! =720

DCD 5040 ;7! = 5040

Value

DCB 5

ALLGN

Result

DCW 0

END

Input and Output of second program

Input: FTABLE = 0001 (0!= 1)

=0001 (1!=1)

=002 (2!=2)

=006(3!=6)

=0018(4!=24)

=0078(5!=120)

=02D0(6!=720)

=13B0(7!=5040)

Value=05

Output Result =0078 (5! = 120)

DATA :Contains data, not instructions. READWRITE is the default

DCW: inserts a byte of padding before the first defined halfword if necessary to achieve two-byte alignment.

If DCB is followed by an instruction, use an ALIGN directive to ensure that the instruction is aligned.

= is a synonym for DCB.

The ALIGN directive aligns the current location to a specified boundary by padding with zeros

Third program optamized factorial

TTL factorial

AREA program,CODE, READONLY

ENTRY

|  |  |
| --- | --- |
|  | ; Assume R0 contains the input value, |
|  | ; and R0 is also containing the return value |
|  |  |
|  | PUSH {R1,LR} |
|  | MOV R1,R0 ; R1 = n |
|  | CMP R0,#1 ; if n>1 recursively call |
|  | BGT next |
|  | MOV R0,#1 ; if n<=1 the return 1 |
|  | B return |
|  | Next |
|  | SUB R0,R0,#1 |
|  | BL fact ; call fact(n-1) |
|  | MUL R0,R0,R1 ; return fact(n-1) \* n |
|  | return |
|  | POP {R1,PC} |

**LR, the Link Register**

Register R14 is used to store the return address from a subroutine. At other times, LR can be used for other purposes.

When a BL or BLX instruction performs a subroutine call, LR is set to the subroutine return address. To perform a subroutine return, copy LR back to the program counter. This is typically done in one of two ways, after entering the subroutine with a BL or BLX instruction:

• Return with a BX LR instruction.

• On subroutine entry, store LR to the stack with an instruction of the form: PUSH {,LR} and use a matching instruction to return: POP {,PC} ...

- push a value (not necessarily stored in a register) means writing it to the stack.

pop means restoring whatever is on top of the stack into a register

\*summary of our presentation\*

1- advantages and disadvantages of assembly language.

2-defintion of assembler like ARMulator assembler and it’s restrictions.

3-we learn some new syntax like DCD,DCB,DCW,ALIGN,SWI.

4-how to build factorial code with just lookup the factorial from a table using the address memory.

5-how to optmize memory in point 4 using stack technique.

Hope you enjoy Dr.Ahmed El nakib

Thank you