Oprea Sergiu-Daniel Gr. 30322 – Lab3

1. Raspundeti la urmatoarele intrebari:
2. Which is the sintax of an instruction?

One instruction occupies one program line and it contains several fields (the brackets indicate that a field can be optional):

[<label>:] [<mnemonic> [<parameter\_1> [,<parameter\_2>]] [;<comment>]

1. What is a label?

- <Label> - symbolic name or identifier given in front of an instruction, can have letters, numbers and special characters.

1. Explain how the following instructions work (what they do, how many parameters,

which restrictions):

1. **MOV** - These instructions transfer data between two registers, one register and a memory location or an immediate value is stored in a register or variable. Memory-to-memory transfers are not allowed and both parameters must have the same size.

MOV instruction

[<label>:] MOV <parameter\_1>, <parameter\_2> [;<comment>]

**LEA** instruction - These instructions allow loading in a register a variable address. The first instruction LEA (Load Effective Address) loads in a register (the first parameter) the offset address of the variable from the 2nd parameter.

1. INC, DEC

Perform increase or decrease of the specified parameter (addition or subtraction with a value of one).

INC <parameter>

DEC <parameter>

1. ADD, SUB, CMP, MUL, DIV

**CMP** instruction - Compares the two operands by performing subtraction, but the result is not stored, only the flags are modified. This instruction is usually followed by a conditional jump instruction.

CMP <parameter\_1>, <parameter\_2>

**ADD** and **ADC** instructions - These 2 instructions perform the addition of 2 operands and store the result in the first operand. The ADC instruction also takes into account the carry flag value.

ADD <parameter\_1>,<parameter\_2>

ADC <parameter\_1>,<parameter\_2>

**SUB** and **SBB** instructions

SUB – subtract the second operand from the first, store the result in the first parameter

SBB – subtract with borrow, also subtracts the carry flag value.

SUB <parameter\_1>,<parameter\_2>

SBB <parameter\_1>,<parameter\_2>

**MUL** and **IMUL** instructions - These instructions perform multiplication for unsigned numbers (MUL) and for signed numbers (IMUL). These instructions have only one parameter because, by default, register EAX is used as the first operand (aka. also stores the result).

MUL <parameter\_2>

IMUL <parameter\_2>

**DIV** and **IDIV** - These instructions perform division for unsigned and signed numbers and, by default, use register EAX for division and result storage.

DIV <parameter\_2>

IDIV <parameter\_2>

1. NOT, AND, OR, XOR, TEST

These instructions implement the basic Boolean algebra, they are performed at bit level and the result is stored in the first operand.

AND, OR, NOT, XOR instructions:

**AND** <parameter\_1>,<parameter\_2>

**OR** <parameter\_1>, <parameter\_2>

**NOT** <parameter\_1>

**XOR** <parameter\_1>,<parameter\_2>

**TEST** instruction

Logical AND between the 2 operands, but the result is not stored. The purpose is to modify the condition indicators and to avoid the destruction of contents from the first operand.

TEST <parameter\_1>,<parameter\_2>

1. PUSH, POP

**PUSH** and **POP** instructions - These 2 instructions work by default with the top level of the stack. PUSH adds an operand on the stack, while POP extract the top level value from the stack and stores is in the specified operand. Please note that the stack for x86 works with sizes of 16 and 32 bit.

PUSH <parameter\_1>

POP <parameter\_1>

1. How do we declare data? Which are the accepted data types?

In assembly language, a variable is a memory location. By declaring a variable, a specific space is reserved in the memory and a symbolic name is attached to the physical address of the variable. The syntax is the following:

<variable\_name> DB|DW|DD|DQ <value1>, [<value2>, <value3>...]

where:

<variable\_name> - symbolic name of the variable

DB – Data Byte – 8 bit size

DW – Data Word – 16 bit size or 2 bytes

DD – Data Double – 32 bit size or 4 bytes

DQ – Data Quadruple – 64 bit size or 8 bytes

<value> - a constant value

We can declare integer variables, real numbers, characters, texts etc.

1. How do we declare constants?

Constants are declared in the same manner as a variable, but some specific rules apply:

<constant\_name> EQU <value>

where:

<constant\_name> - symbolic name of the constant

EQU – mnemonic for constants ("equals")

<value> - a constant value Cum se declara segmentele?

1. How do we declare segments?

A segmented memory model divides the system memory into groups of independent segments referenced by pointers located in the segment registers. Each segment is used to contain a specific type of data. One segment is used to contain instruction codes, another segment stores the data elements, and a third segment keeps the program stack.Syntax

name SEGMENT ⟦READONLY⟧ ⟦align⟧ ⟦combine⟧ ⟦use⟧ ⟦characteristics⟧ ALIAS(string) ⟦'class'⟧

statements

name ENDS

1. How do we declare procedures?

Procedures or subroutines are very important in assembly language, as the assembly language programs tend to be large in size. Procedures are identified by a name. Following this name, the body of the procedure is described which performs a well-defined job. End of the procedure is indicated by a return statement.

Syntax

Following is the syntax to define a procedure −

proc\_name:

procedure body

...

ret

The procedure is called from another function by using the CALL instruction. The CALL instruction should have the name of the called procedure as an argument as shown below −

CALL proc\_name

1. What does ASSUME directive mean?

The ASSUME directive shows the segment name to the assembler. It provides information to the assembler regarding the name of the program or data segment for that particular segment.This "binding" is useful to automate some common patterns:

It tell the assembler which segment register to use to access a variable.

If you don't explicit a segment register during a memory access the assembler uses the ASSUMEd values to automatically add a segment override prefix to the instruction.

If a segment has not been ASSUMEd to be pointed by any segment register, the assembler fails with an error if you try to load/store a variable inside that register.

It tell the assembler with respect to which segment to calculate the offsets.

If you specify a segment register in a memory access the assembler use the segment ASSUMEd for that segment register to calculate the offset of the memory access.

Note that although DS is implicitly used by the CPU on every memory access, an explicit override with DS: is needed make clear the intention to use its segment as a base for the offset.

The segment ASSUMEd by CS is the segment the code labels belong to.

You can't jump/call a symbol unless it is in the segment ASSUMEd by CS.

1. What does END directive mean?

The END directive informs the assembler that it has reached the end of a source file.

Syntax

END

Usage

Every assembly language source file must end with END on a line by itself.

If the source file has been included in a parent file by a GET directive, the assembler returns to the parent file and continues assembly at the first line following the GET directive.

If END is reached in the top-level source file during the first pass without any errors, the second pass begins.

If END is reached in the top-level source file during the second pass, the assembler finishes the assembly and writes the appropriate output.