# Instruction Set Architecture

## Simple Processor model.

- En processor har ett X antal av General Purpose Registers.
- Word Length är 32 bits (4 bytes)
- Memory är byte addressable.
- Varje instruction är one word lång.
- Instructions tillåter one memory operand pär instruction.
    - One register operand tillåts också med en memory operand.

## Conditional flags

The processor keeps track of the information about the results of previous operation. Information is recorded in idividual bits called **“Condition code flags”**. There is some common flash which are.

- N (Negative, set to 1 if result is negative, else cleared to 0)
- Z ( Zero, set to 1 if result is zero, else cleared to 0)
- V ( Overflow, set to 1 if arithmetic overflow occurs, else cleared)
- C (Carry, set to 1 if a carry-out results, else cleared)

**Condition code flags** are grouped together in a special purpose register called **“Conditon code register”** or **“status register”.**

Ex. If the results of a decrement r1 in the case that r1 is 1 and becomes 0 Z is set. And if you call branch >0 it will test the Z flag which in this case is 1.

## Branch.

- **By using branch we alter the sequence of the program execution**
    - PC holds the address of the next instruction to be exectuted.
    - If we branch we load a new value into PC.
    - Processor fetches & exectures the instruction at this new adress instead.
    - The new address is called “**branch target**”.
- **A conditonal branch only affects PC if the condition is met.**
    - Uses condition code flags to check if conditions are met.

## Adress

- **move #Num1, r2 (Initialize r2 with the address of num1)**

## Stacks

- A stack is a list of data elements, usually words or bytes with the accessing restriction that elements can be added or removed at one end of the stack.
    - The end from which elements are added and removed is called the **“top”** of the stack.
    - Other end is called **“bottom”**
- Other names for stack is
    - Pushdown stack
    - Last in first out (LIFO) stack
- **Push -** placing a new item on the stakc
- **Pop -** removing the top item from the stack.
- **Data stored** in the memory of a computer can be oranized as a **stack**
    - Successive elements occupy successive memory locations.

- New elemnts which are **pushed** into the **stack** ar eplaces in successively lower address locations
    - A stack grows in direction of decreasing memory addesses.
- A processor register called as **"Stack pointer (SP)"** is used to keep track of the address of the element that is at the top at any given time.
    - A general purpose register could serve as a stack pointer.

**Subroutines**

- Subtasks that are repeated on different data values are usually implemented as **subroutines.**
- The use of **subroutines** branches to the subroutine.
    - Branching to a subroutine is called **"calling"**
    - Instruction to perform this is called **"Call"**
- After the execution of a subroutine is finished the program continues after the "**Call"** instruction.
    - A subroutine "Returns" the program to the point in which it was called
    - The instruction for this is called **"Return"**
- The subroutines know where to return to since the PC s value which was the next instruction before the CALL was called is stored by the **call** instruction.
- The way in which a processor makes it possible to call and return form a subroutine is called **"subroutine linkage method".**
- The **return** address could be stored in a register called as "**Link register"**
- **Call** instruction
    - Stores contents of the PC in a link reigster
    - Branches to the subroutine.
- **Return** instruction
    - Branch to the address contained in the link register.

- Nested subroutines
    - Push the return addresses onto a stack as they are generated by subroutine calls.
    - Pop the return addresses from the stack as they are needed to execute return instructions.

**Assembly**

- Instead of using real words when we program we use acronyms called **Mnemonics**.
    - Example MOVE A TO BE = mov a,b
- Programs written in assembly needs o be translated into a understandable form by the computer, **binary or machine language.**
- This task is performed by an **assembler**
    - Assembly source code is called **source program**
    - Assembled machine language program is called **object program**
- Each **mnemonic** represents the **binary pattern, or opcode** for the operation performed by the instruction.
- **Assembly language** must have a way to indicate the addressing mode being used for the operand addresses.

- The statement which provides additional information to the assembler to translate source program into an **object program** are called **assembler directives or commands.**
- Assembly instructions have a generic form
    - **Label Operation Operand(S) comment**
- Four fields are separated by a delimiter, typically one or more blank characters.
- Label is optionally associated with **a memory address:**
    - May indicate the address of an **instruction** to be executed.
    - May indicate the address of a data item.

**Encoding of machine instructions**

- Instruction specify the **operation** to be performed and the **operand** to be used.
- Which operation is to be performed and the addressing mode of the operands may be specified using an **encoded binary pattern** referred to as the **"OP Code"** for the instruction.
- Read slides 23 - 28 Pictures explain more than text can.