Why study the Alpha (assembly) ?

The Alpha architecture is the first 64-bit load/store RISC (as opposed to CISC) architecture designed to enhance computer performance by improving clock speeding, multiple instruction issuing and multiple processors management.

- RISC (Reduced Instruction Set Computer) processors are based on the fact that working with internal registers is faster (orders of magnitude) than accessing memory via Bus.
- Intended (in the 80's) as a 25 years long project
    - First 21st century computer architecture

Alpha RISC true 64-bit load/store architecture:

- Use of internal registers which can be loaded from and results transfer back into memory.
- Other work is done between registers
- Instructions interact with each other only by one instruction writing a register or memory and another instruction reading from the same place
    - Can issue multiple instructions every CPU cycle
    - Up to 12 instructions per cycle today

The alpha 21164 was the first processor to break the billion instructions per second threshold in 1996
Alpha processors leader in performance in the 64-bit architecture area
Compaq has announced technology merge with Intel

- Standardisation of future 64-bit architecture will include architectural features of the alpha processors

What is a register?
An 8 bytes space referred to by a name (you will see the different conventions later on) that holds the current processor state.

Alpha Architecture

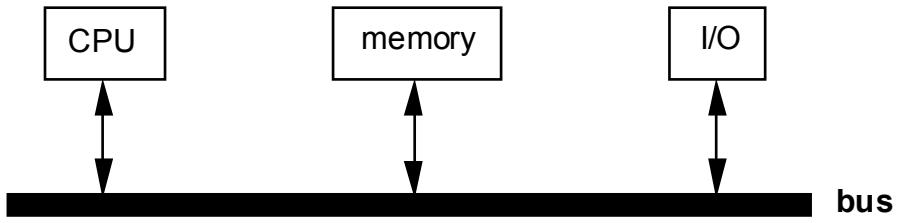
A computer consists of three main components, CPU, memory and I/O devices. The components are connected by a bus, which is used to transmit information amongst the components.

CPU (Central Processing Unit): CPU is used to carry out computation.

Memory: The memory module stores data and programs.

- RAM, ROM, data storage
- Alpha: byte addressable memory
  - Each memory location has an address and contents
  - Memory is a list: the further down the list goes, the higher the address is.

I/O devices: The I/O devices (monitors, keyboards, printers) act as the interface between the computer and



the users.

In this part of the course, we are only interested in the CPU of a computer and how it operates.

Alpha chip.

Alpha was designed by Digital as a 64-bit (load and store) architecture.

- All registers are 64 bits in length.
- All operations (data manipulation) are performed between registers.
- External access are only memory operations such as loads or stores

Instructions are coded as 32 bits (longword / 4 bytes) sequences.
Is it confusing?
Example:

addq Register1, Register2, Register3 <-- **This is assembly :)**

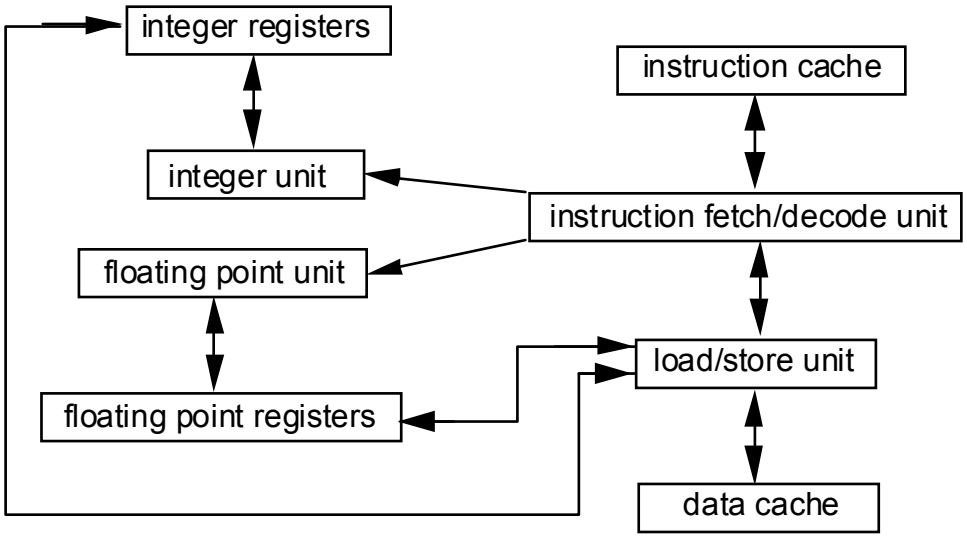
Stands for the addition between what is in Register 1 and r
Register 2 with the results being contained by Register 3.
Addq stands for addition between quadwords (e.g 64 bits long number).
Therefore each register contains a quadword usually displayed in its hexadecimal form:

Register1 contents=0x123456789abcdef0 just an example

The instruction has to be coded to be interpreted and executed by the processor.
It is coded as 32-bit long value which will contain identification code (usually called opcode) of the operation performed and information on which registers are involved.

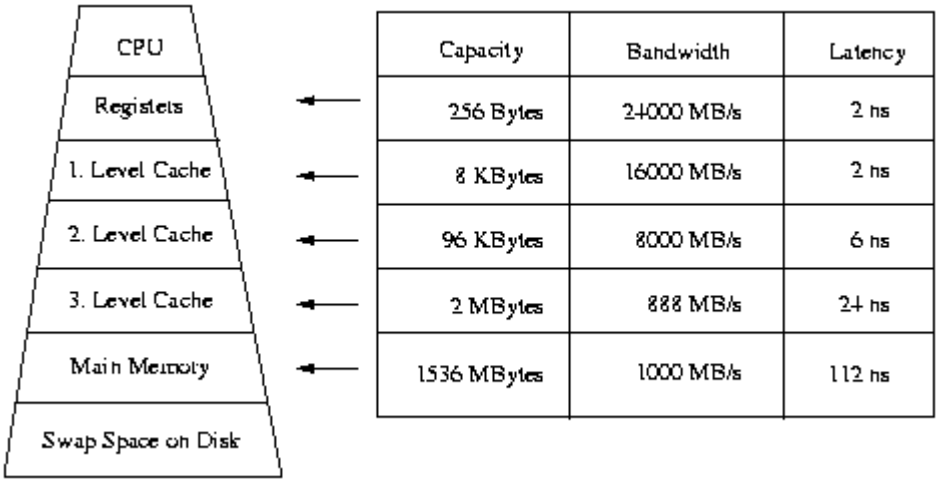
The above instruction is coded as *0x40220403*

Simplified Functional block diagram of the 21164 Alpha chip



Features of the 21164 Alpha chip

- 64 bit RISC architecture
- 32 integer registers
- 32 entry, 64 bit floating-point register file
- 8 KB, direct-mapped, L1 instruction cache (onchip)
- 8 KB, direct-mapped, write through L1 data cache (onchip)
- 96 KB, L2 data and instruction cache (onchip)



Memory hierarchy

ALU Units:
Arithmetic/Logic Unit where arithmetic and logical operations of the CPU are performed. Different units are in charge of integer, floating points and address operations.

Integer unit: This unit carries out arithmetic (addition, subtraction) and logic (shift left or right) operations on integers.

Integer registers: Integer registers store the integers used during operations

Instruction cache: The instruction cache stores the next sets of instructions to be executed by the CPU.

Data cache: Data cache holds the values of the variables in the programs.

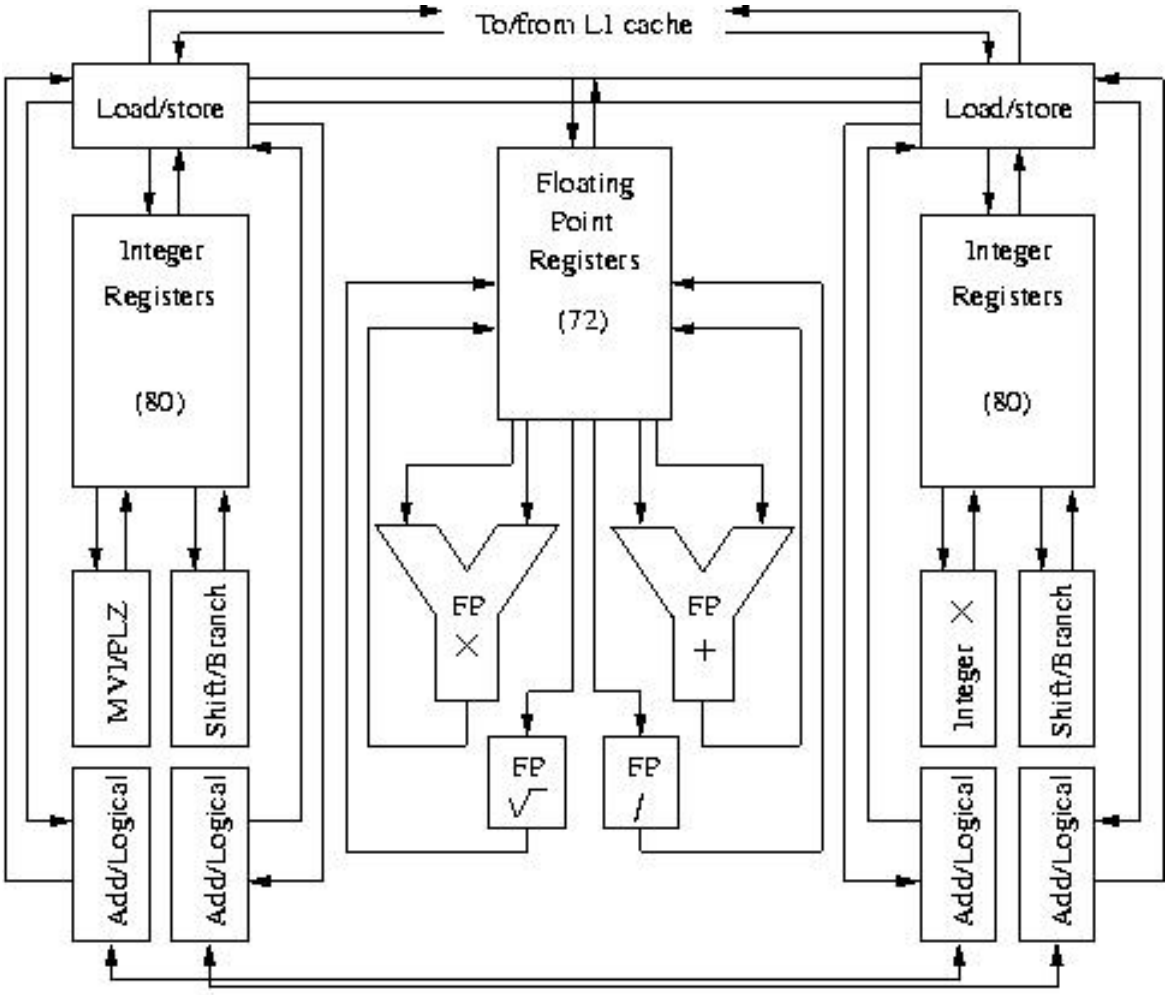
- Local memory, which may be addressed several times faster than the primary memory. Used to speed up time-consuming tasks such as loading/unloading instructions/data from memory.
- The bigger, the better (16 KB -> 112 KB so far)

Load/store unit: This unit is responsible for controlling the data transfer between the data cache and the register.

Instruction fetch/decode unit (control unit):

- This unit fetches the instructions from the instruction cache.
- The instructions are interpreted; and, according to the instructions, the unit issues signals to the other components in the CPU to control their operations.
- Program counter (PC) register is in this unit. The PC stores the address of the next instruction to be executed. The unit fetches the next instruction according to this address. After an instruction is fetched, the value of the PC is updated to point to the next instruction in the memory.

Functionnal diagram of an alpha EV-7 processor



(a)

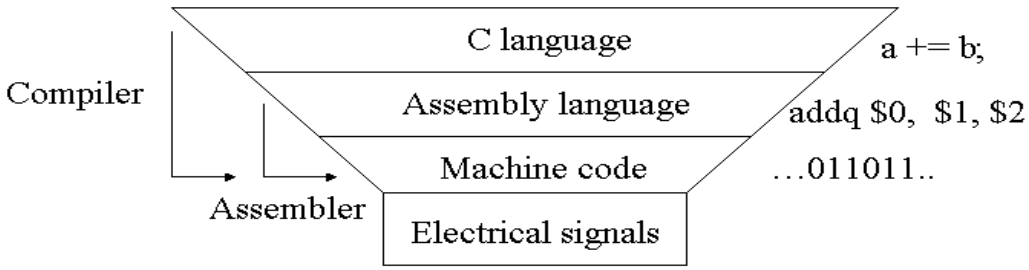
Features:

- *Two* duplicate integer register units.
- Four integer Add/Logical units (exchange values in one cycle if required).
- Two load/store units draw on a 64 KB instruction and a 64 KB data cache.
- 80 integer and 72 floating-point registers.
- 4 dual channels (North, East, South, West) from the chip to interconnect it with neighbouring chips at a bandwidth of 1.6 GB/s per single channel.
- direct I/O dual link from the chip has a bandwidth of 1.6 GB/s
- The chip is expected to ship first at a clock frequency of 1-1.2 GHz.

Why study assembly?

"Every Computer Scientists should master Architecture-Algorithms-Applications (AAA)"

From high-level language to low-level language:



Example:
To do a = a + b I can write in C (C++):

- a += b;

In our alpha assembly, it is written as:

- 0x40014400 (addq $0, $1, $0)

In Binary machine code it will look like:

- ..01000000000000100000100000000010…

Machine language

- Flow of 0's and 1's
- Higher-level languages have to be transformed (via a compiler) to it before a program can be run by the computer

Assembly language:

- mnemonics are used to represent the machine instructions (*e.g it is easier to write (and remember) addq to perform the addition between 2 quadwords with the result as a quadword rather than having to provide and hexadecimal code or a set of 0s and 1s*)
- Assembler transforms the assembly code into the machine code

- Unlike high-level languages, assembly languages are not portable between platforms using different architectures.
- This is because instructions in an assembly language are tied to the instruction set of a particular type of CPU.
- Different types of CPUs have different instruction sets.
    - Instruction set: the instructions, which can be understood by the machine.
- Harder to debug (not meant to be done by "normal" humans anymore).
- Needs to reconstruct algorithm architecture
    - *Instructions might have to be modified, orders may differ (we will see examples later on)*
- Time consuming to implement and it is very easy to make mistakes.Fast: takes the maximum from the CPU architecture as it is designed for it.
- Widely use for multimedia applications requiring real time processing.
- Applications that have to run nearly at the limit of the machine's performance, typically graphics-oriented games.
- Applications that must run on a given system
- Embedded applications (main cost: hardware)For the future: bionics, robotics, etc…

Goal:

- Obtain a better understanding of the underlying computer structure that makes your programs work. We are going to learn the Alpha assembly language. The CPUs of "cs26" and "kahu" are Alpha chips. We could use the assembler on these two servers to generate the machine code and test your programs.
- There are not any tools on cs26 for us to observe the execution of our programs, e.g. set break point, and see the value of the registers.
- This makes debugging very difficult
- Machine might crash each time the programs are not working…

Instead

Use an Alpha simulator written by Dr. Bruce Hutton. The simulator has a very good user interface and a set of very useful tools for observing and debugging programs.

- Only a simplified version of alpha assembly but it will allow you to understand how instructions are processed, memory allocated, register modified, local variables defined, etc….