What you should know by today

CPU basic architecture
- RISC, Units, Local Memory, Registers, Instructions

Registers
- Number, size, kind

Instructions
- Size, kind

Data representation
- Hexadecimal, little-big endian, one’s and 2’s representation and computation rules.

Useful web address to find support on assembly, simulator

Tutorials:
- Architecture, registers, instructions format
- Using the simulator

Demonstrator:
- Familiar with the alpha simulator

The Layout of an Assembly Language Program

Example 1:

What does an assembly language program look like?

```
entry main.enter;

import "../IMPORT/register.h";
import "../IMPORT/callsys.h";

//      void main()
//              {
//              while ( TRUE )
//                      {
//                      char c;
//                      c = getChar();
//                      putchar( c );
//                      }
//              }
block main uses register, CALLSYS {
        code {
        public enter:
        loop:
                ldiq $a0, CALLSYS_GETCHAR;
                call_pal CALL_PAL_CALLSYS;
                mov $v0, $a1;
                ldiq $a0, CALLSYS_PUTCHAR;
                call_pal CALL_PAL_CALLSYS;
                br loop;
        end:
        }
}
```

Homegrown assembler.

The program reads characters from the keyboard, and outputs them to the screen.

- The line `entry main.enter;` specifies the entry point for the program (where the program starts executing).
- The lines

```
import "../IMPORT/register.h";
import "../IMPORT/callsys.h";
```

specify that the code in the specified files is imported (included).

The C-like code, with `//` to the left of each line, is a sequence of comments.
Assembler ignores them.

Assembly language programs are very low level and difficult to read

- MUST document your assembly language program by comments written in a high level language or/and describing precisely what registers are used for and when.

The lines
block main uses register, CALLSYS {
}
specify that we are creating a block of code called main, using some definitions in a block called register (that specifies the register numbers for the symbolic names a0, a1, v0, etc,…) and also using some definitions in a block called CALLSYS (that specifies the values of CALL_PAL_CALLSYS, CALLSYS_GETCHAR, CALLSYS_PUTCHAR, etc).
The lines
code {
}
just specify that we are defining code (instructions), rather than data.

The line *public enter:*
labels some code, with the name “ enter”. This code can be referred to outside the block, as *main. enter*.

The lines:

```
loop:
ldiq $a0, CALLSYS_GETCHAR;
call_pal CALL_PAL_CALLSYS;
mov $v0, $a1;
ldiq $a0, CALLSYS_PUTCHAR;
call_pal CALL_PAL_CALLSYS;
br loop;
end:
```

represent the real work.

The identifier “loop” labels the beginning of the loop.

- As in all computer languages, the name is arbitrary
- Could be replaced by any other identifier.

The line *ldiq $a0, CALLSYS_GETCHAR;* loads the constant value CALLSYS_GETCHAR (whatever that has been defined to be) into the register a0.

The line *call_pal CALL_PAL_CALLSYS;* makes a request (rather like a function invocation) to the operating system to do something (read a character from the keyboard). The operating system uses the value in register a0 (**by convention**) to determine what action to perform (in this case read a character), and returns the result in register v0 (**by convention**). Other parameters to system calls may be passed in registers a1, a2, a3, ...

Register.h

Contains declaration of symbolic names for the registers

```
block register {
        abs {
        public v0    =     0;
        public t0    =     1;
        public t1    =     2;
        public t2    =     3;
        public t3    =     4;
        public t4    =     5;
        public t5    =     6;
        public t6    =     7;
        public t7    =     8;
        public s0    =     9;
        public s1    =     10;
        public s2    =     11;
        public s3    =     12;
        public s4    =     13;
        public s5    =     14;
        public s6    =     15;
        public fp    =     15;
        public a0    =     16;
        public a1    =     17;
        public a2    =     18;
        public a3    =     19;
        public a4    =     20;
        public a5    =     21;
        public t8    =     22;
        public t9    =     23;
        public t10   =     24;
        public t11   =     25;
        public ra    =     26;
        public pv    =     27;
        public at    =     28;
        public gp    =     29;
        public sp    =     30;
        public zero  =     31;
                }
        }
```

Callsys.h

Contains opcode values corresponding to functions, which will generate special actions when request to operating systems is made.

```
block CALLSYS {
        abs {
                public CALL_PAL_CALLSYS         =       0x83;
                public CALLSYS_EXIT             =       0x0;
                public CALLSYS_GETCHAR          =       0x1;
                public CALLSYS_PUTCHAR          =       0x2;
                public CALLSYS_FORK             =       0x3;
                public CALLSYS_EXEC             =       0x4;
                public CALLSYS_GETINPUTFILE     =       0x5;
                public CALLSYS_WAIT             =       0x6;
                public CALLSYS_MAX              =       0x7;
                }
        }
```

PALcode (Privileged Architecture Library):

A set of subroutines that is specific to a particular Alpha operating system implementation (Unix, Windows NT). These subroutines provide operating-system primitives for context switching, interrupts, exceptions, and memory management. PALcode is similar to the BIOS libraries that are provided in personal computers.
PALcode subroutines are invoked by implementation hardware or by software **CALL_PAL** instructions.