UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

CROSS DEVELOPMENT FOR EMBEDDED SYSTEMS

BEESHANGA ABEWARDANA JAYAWICKRAMA

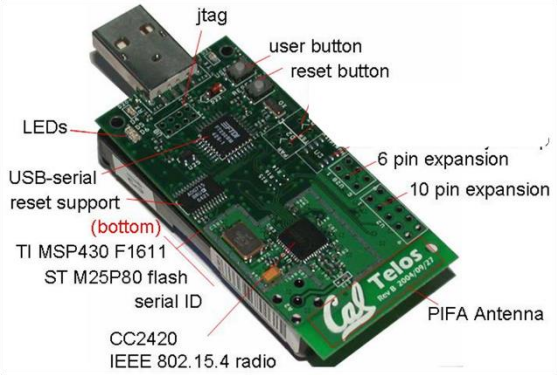
UTS:
ENGINEERING AND
INFORMATION
TECHNOLOGY

feit.uts.edu.au

CROSS DEVELOPMENT

What is "Cross Development"?

> Creating programs on one computer for execution on another

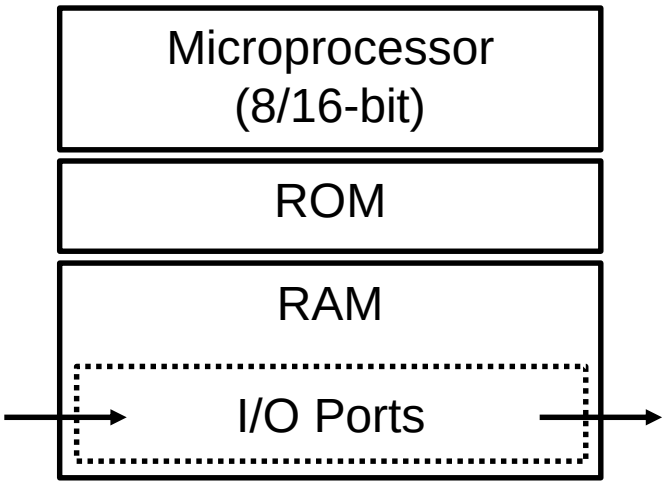


Development "Host"
PC or Unix workstation with cross-development tools installed, including a communications program

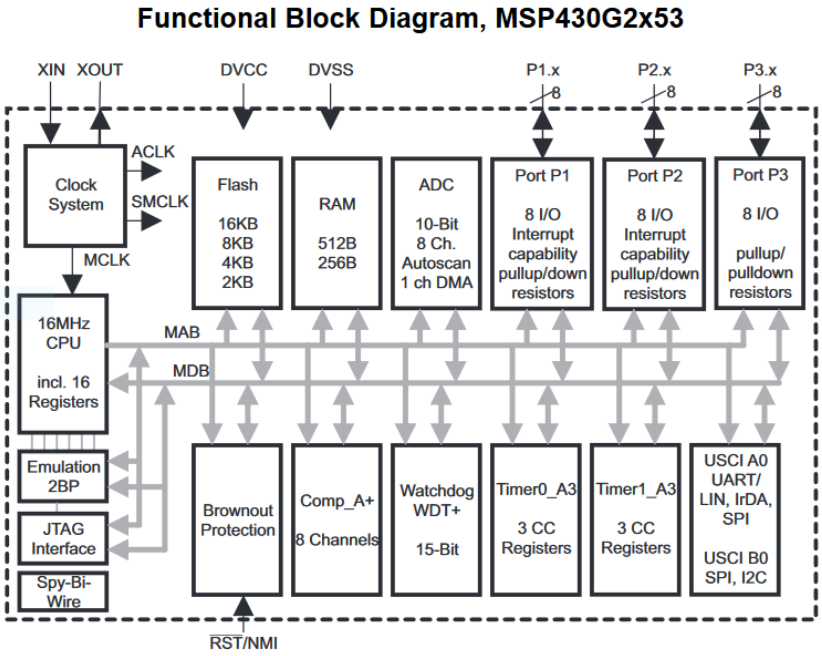
Development "Target"
Embedded computer with processor, ROM, RAM and I/O devices and communications ability

MICROPROCESSOR ARCHITECTURE

Generic microprocessor architecture



TI MSP430 16-bit microprocessor architecture



WHY CROSS DEVELOP?

Develop on a computer and transfer and run the executable on a TelosB microcontroller.

Why not develop directly on TelosB?

> TelosB platform does not support a monitor

> It does not have a hard disk that can store a large volume of data

> Processor speed is only sufficient for very specific repetitive tasks

Where else would we cross develop?

A programmer can develop an app in Linux, cross compile the program to run on Windows/Mac.

> No need to own a Windows/Mac (most of the time).

CROSS COMPILATION

A compiler that generates assembly language for a different processor than the one the compiler is running on is called a **Cross Compiler**, as opposed to a **Native Compiler**

Native compilation with GCC:

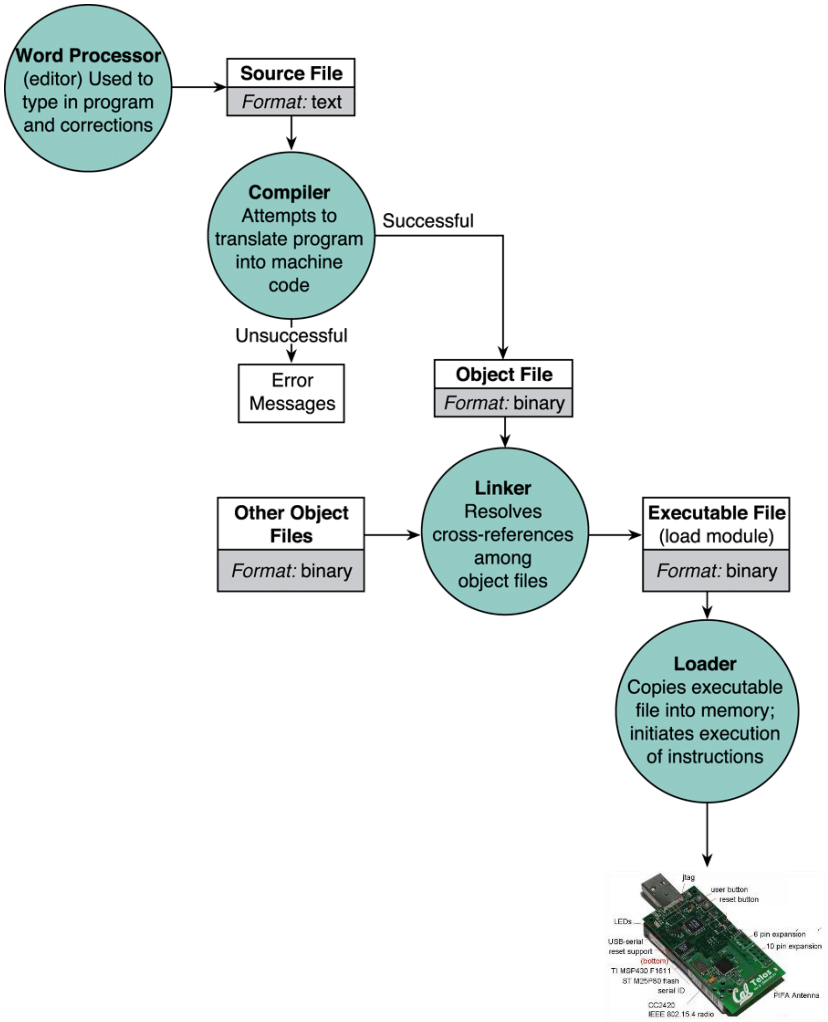
```
gcc -Wall -ansi -lm my.c -o my.out
```

Cross compilation with GCC:

```
gcc -Wall -ansi -lm --target=some-target my.c -o my.out
```

The host must know the target platform (CPU, memory and operating system) and have target platform's C standard library – configure binutils in GCC

COMPILING, LINKING AND LOADING



Cross compiler

> Must know how to generate instruction for target processor

Cross linker

> Must know about the memory and device layout of the target system so that it can set the appropriate starting addresses

Loader

> A communications tool is needed to transport the binary after linking

Debugging might not be supported at all

CROSS DEVELOPMENT FOR EMBEDDED SYSTEMS

Standard C libraries need to be modified to fit the target system

> E.g. printf() for a PC monitor, might not be the same as printf() for a LCD

Operating systems may be basic or not present

> Need to suit its memory/port mapped I/O interfaces and microprocessor instructions

Transferring of the program relies on firmware on device's ROM

> Connection could be made with serial, infrared or Ethernet ports.

UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

COMMAND LINE PARSING

BEESHANGA ABEWARDANA JAYAWICKRAMA

UTS:
ENGINEERING AND
INFORMATION
TECHNOLOGY

COMMAND LINE PARSING

Allows to pass parameters to a program through the command line at the time of execution.

E.g. `gcc -Wall -ansi -lm my.c -o my.out`

GCC is a program on the computer. All other are arguments parsed through the command line before the program is started.

The shell (command interpreter) gathers up the strings and gives them to the program.

Perhaps easier to execute programs this way, rather than taking standard input after the program has been started to run.

WRITING THE MAIN FUNCTION TO ACCEPT COMMAND LINE ARGUMENTS

The program is only able to accept the strings if it is told to expect them. A special variation on the main function prototype is necessary for this to happen:

```
int main(int argc, char* argv[]);
```

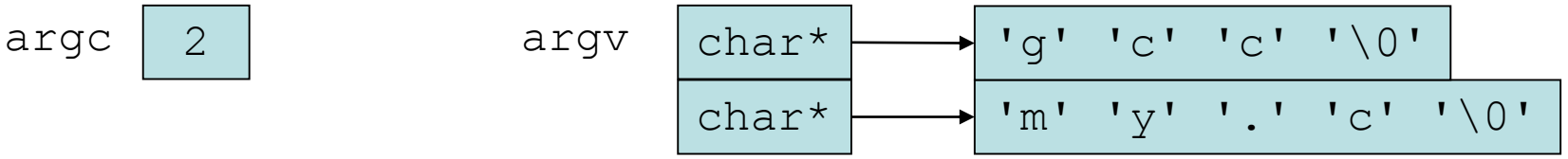
`int argc` is a count of the number of separate strings that have been typed on the command line including the program name itself

`char* argv[]` is an array of char pointers that holds all the strings including the program name itself

These variable names are used by convention

ARGC AND ARGV EXAMPLE

```
gcc my.c
```



`argv` is an array of size 2 that contains char pointers. Each char pointer points to an input argument string.

Arguments can be accessed within the main function as follows: