National Technical University of Athens
Microprocessors and Digital Systems Laboratory

ARM Assembly examples

Masouros Dimosthenis
Manolis Katsaragakis
Ioannis Oroutzoglou
Aggelos Ferikoglou

To begin with…

- Start QEMU emulator
- Write and compile a simple “hello_world.c”
  - gcc -Wall hello_world.c -o hello_world
  - file hello_world
- Extract the assembly out of the C source code
  - gcc -Wall hello_world.c -S
- Compile assembly:
  - gcc –Wall hello_world.s -o hello_world
- Thoughts?

ARM assembly template

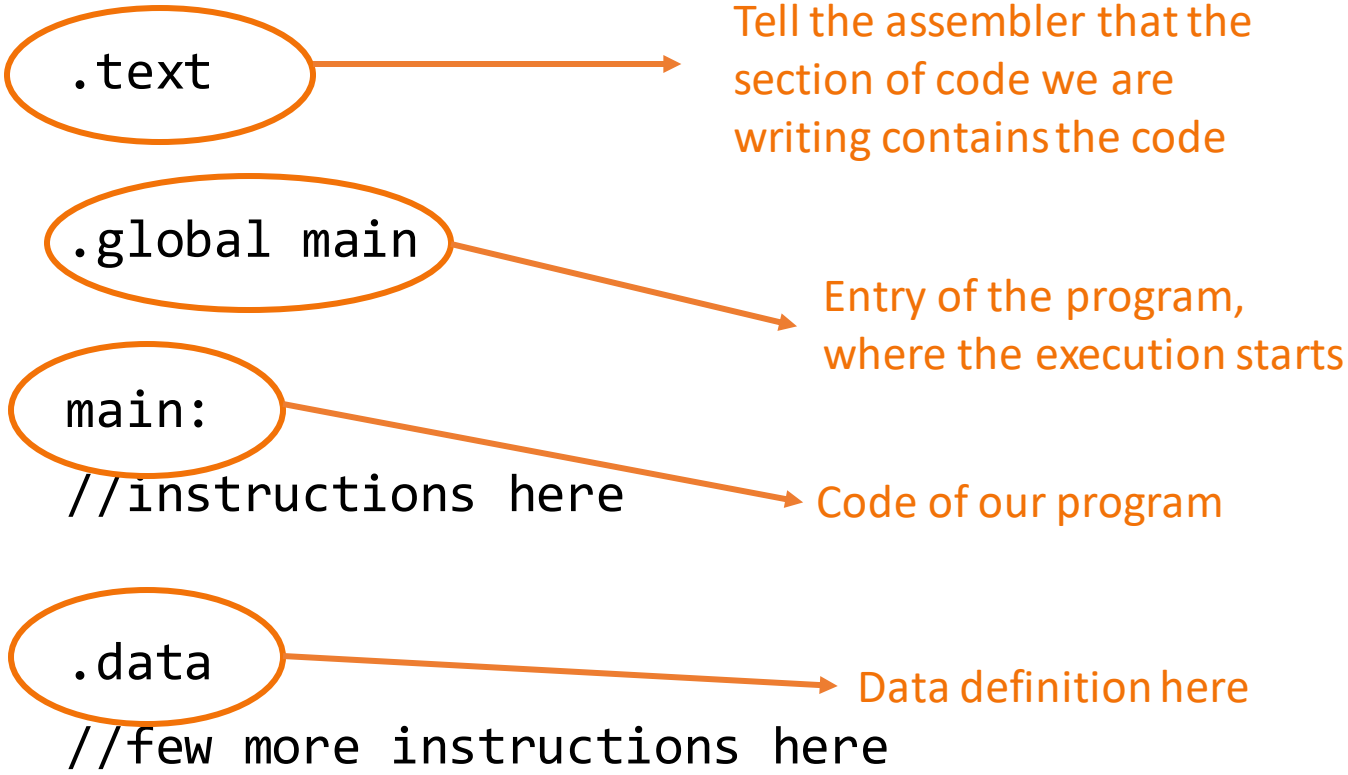
```
.text

.global main

main:
//instructions here

.data
//few more instructions here
```

ARM assembly template



First program in ARM assembly

C code

```
#include <stdio.h>

int main(){
    int a=15;
    printf("Number=%d\n,a);
}
```

ARM assembly code

- Things to consider
  - How do I call the printf function?
  - How do I define the message string?

First program in ARM assembly

C code

```
#include <stdio.h>

int main(){
    int a=15;
    printf("Number=%d\n,a);
}
```

ARM assembly code

```
.text
.global main
.extern printf

main:
    push {ip,lr}
    ldr r0, =string
    mov r1, #15
    bl printf
    pop {ip,pc}

.data
string: asciz "Number=%d\n"
```

- The arguments of printf are given to registers r0,r1 in the same order
- This is a general rule for calling external functions

GCD challenge

C code

```
#include <stdio.h>

void main(){
    int a=53;
    int b=43;
    while (a!=b)
    {
        if (a>b)
            a=a-b
        else
            b=b-a
    }
    printf("GCD=%d\n",a);
}
```

ARM assembly code

```
.text
.global main
.extern printf
main:
    mov r3,#53
    mov r4,#43
gcd:
    //your code here
    push {ip,lr}
    ldr r0, =string
    mov r1,r3
    bl printf
    pop {ip,pc}

.data
    string: .asciz "GCD=%d\n"
```

- Compile and run it
- Check produced assembly
  - Use -O0 and -O3 as compile arguments
- Can you do it better?

REMEMBER: Embedded systems is about efficiency

GCD challenge: The winner

C code

```
#include <stdio.h>

void main(){
    int a=53;
    int b=43;
    while (a!=b)
    {
        if (a>b)
            a=a-b
        else
            b=b-a
    }
    printf("GCD=%d\n,a);
}
```

ARM assembly code

```
.text
.global main
.extern printf
main:
    mov r3,#53
    mov r4,#43
gcd:
    cmp r3,r4
    subgt r3,r3,r4
    suble r4,r4,r3
    bne gcd
    push {ip,lr}
    ldr r0, =string
    mov r1,r3
    bl printf
    pop {ip,pc}

.data
    string: .asciz "GCD=%d\n"
```

- Compile and run it
- Check produced assembly
  - Use -O0 and -O3 as compile arguments
- Can you do it better?

REMEMBER: Embedded systems is about efficiency

Multiplication

- Suppose r2=3 and r3=27
- Write two assembly programs that implement the arithmetic operation r2*r3 and print the result
  - You can use any command
  - You cannot use MUL/MLA commands

TIP: Remember the barrel shifter?

ARM assembly code

Multiplication

- Suppose r2=3 and r3=27
- Write two assembly programs that implement the arithmetic operation r2*r3 and print the result
  - You can use any command
  - You cannot use MUL/MLA commands

TIP: Remember the barrel shifter?

ARM assembly code

```
.text
.global main
.extern printf
main:
    mov r2,#3
    mov r3,#27
    add r4,r2,r2,LSL#3
    add r5,r4,r4,LSL#1
    push {ip,lr}
    ldr r0, =string
    mov r1,r5
    bl printf
    pop {ip,pc}

.data
    string: .asciz “Number=%d\n”
```

Linux system calls

- **What is a system call?**

Call in which program requests a service from the kernel of the operating system it is executed on.

- Jumps from user space to kernel space

```
#include <unistd.h>
```

- `int open(const char *pathname, int flags);`
- `ssize_t read(int fd, void *buf, size_t count);`
- `ssize_t write(int fd, const void *buf, size_t count);`
- `int close(int fd);`
- `void _exit(int status);`

unistd.h of ARM debian

- Each system call is associated with a number-offset starting from a base (__NR_SYSCALL_BASE + 0/1/2/3/…)

```
#if defined(__thumb__) || defined(__ARM_EABI__)
#define __NR_SYSCALL_BASE 0
#else
#define __NR_SYSCALL_BASE __NR_OABI_SYSCALL_BASE
#endif

#define __NR_restart_syscall (__NR_SYSCALL_BASE+ 0)
#define __NR_exit (__NR_SYSCALL_BASE+ 1)
#define __NR_fork (__NR_SYSCALL_BASE+ 2)
#define __NR_read (__NR_SYSCALL_BASE+ 3)
#define __NR_write (__NR_SYSCALL_BASE+ 4)
#define __NR_open (__NR_SYSCALL_BASE+ 5)
#define __NR_close (__NR_SYSCALL_BASE+ 6)
```

System Calls – calling convention

- :~$ man syscalls
- syscall number at r7
- Arg1 to r0, Arg2 to r1, … , Arg7 to r6
- Return value to r0
- After everything is set, execute swi 0
- Software interrupt!

Questions?