F28HS Hardware-Software Interface

Lecture 10: ARM Assembly Language 5

Software interrupt

`SWI` *operand*

– *operand* is interrupt number

- halts program
- saves PC
- branches to interrupt service code corresponding to *operand*

Software interrupt

- Linux/Posix provides an API to core system functions
  - file system, process control etc
- based on table of addresses of functions
- access via software interrupt

R7 == system function id

R0 - R6 == arguments

- after interrupt, hardware calls function in position R7 of table
- result returned in R0
- really fast mechanism!

File read

- read
  - R7 == 3
  - R0 == file descriptor - keyboard == 0
  - R1 == address for byte sequence
  - R2 == number of bytes to read
  - returns a count of chars read in R0 or 0 at EOF

File write

- write
  - R7 == 4
  - R0 == file descriptor - monitor == 1
  - R1 == address of byte sequence
  - R2 == number of bytes to write

Example - copy keyboard to screen

```
.global _start
_start:
loop:   MOV R0, #0
        LDR R1, =char
        MOV R2, #1
        BL read
        MOV R0, #1
        LDR R1, =char
        MOV R2, #1
        BL write
        B loop
```

```
read:   MOV R7, #3
        SWI 0
        BX LR

write:  MOV R7, #4
        SWI 0
        BX LR
…
.data
char: .word 0x00
```

Example - print decimal

- build table of powers of 10
- for each power of 10 from 10^N to 0
  - divide value by power of 10
    - subtract & count
  - add ‘0’ to count
  - print count as character
- 2^32 == 4294967296 == 10 digits
- need 1st 10 powers of 10

Example - make 10^N table

R0 == next 10^N addr

R1 == count

R2 == next 10^N

R3 == 10

```
tens:   .rept 10
        .word 0x00
        .endr
...
```

```
make10s:LDR R0, =tens
        MOV R1, #10
        MOV R2, #1
        MOV R3, #10
...
```

Example - make 10^N table

```
...
pow10:  STR R2, [R0]@ store next 10^N
        SUB R1, #1  @ decrement count
        CMP R1, #0  @ finished?
        BEQ done10
        ADD R0, #4  @ increment 10^N addr
        MUL R2, R3  @ make next 10^N
        B pow10
done10: BX LR
```

Example - print decimal

R0 == value

R1 == count of 10^N

R2 == address of next 10^N

R3 == next 10^N

R4 == division count

```
printd: MOV R1, #10
        LDR R2, =tens
        ADD R2, #36
...
```

Example - print decimal

```
...
nextchar:
        LDR R3, [R2] @ get next 10^N
        MOV R4, #0   @ division count = 0
loop10: CMP R0,R3    @ finished division?
        BLT showd
        SUB R0, R3   @ take away 10^N
        ADD R4, #1   @ increment count
        B loop10
...
```

Example - print decimal

```
showd:  ADD R4, #'0`  @ count -> char
        LDR R5, =char @ store char
        STR R4, [R5]
        PUSH {R0}     @ save R0-R2
        PUSH {R1}
        PUSH {R2}
        MOV R0, #1
        LDR R1, =char
        MOV R2, #1
        PUSH {LR}     @ save LR
        BL write
```

Example - print decimal

```
...
         POP {LR}      @ restore LR
         POP {R2}      @ restore R0-R2
         POP {R1}
         POP {R0}
next10:  SUB R1, #1    @ decrement 10^N count
         CMP R1, #0    @ finished?
         BEQ endp
         SUB R2, #4    @ decrement 10^N addr
         B nextchar
endp:    BX LR
```

Command line arguments

- passed on stack

*SP == `argc`

*(SP+4) == address of `argv[0]` == name of executable

*(SP+8) == address of `argv[1]` == 1st argument

*(SP+12) == address of `argv[2]` == 2nd argument

etc

Address offset notation

- can specify offset from register in address operand
- e.g. in `LDR`, `STR` etc

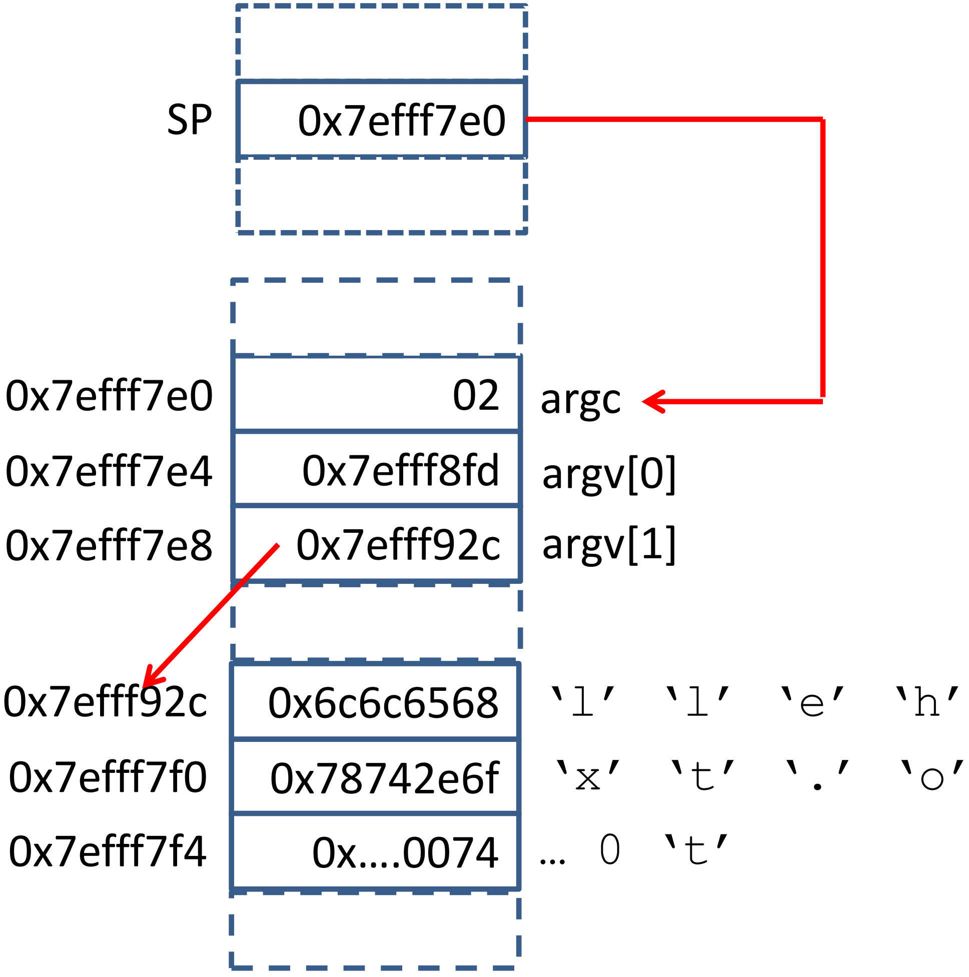
`LDR` R_i, `[`R_j, `#`*int*`]` ==

`ADD` R_j, `#`*int*

`LDR` R_i, `[`R_j`]`

Example - show argv[1]

```
.global _start
_start:
        LDR R1, [SP,#8]
        LDR R2, =argv1
        STR R1, [R2]
        BL length
        MOV R0, #1
        LDR R3, =argv1
        LDR R1, [R3]
        BL write
        BL _exit
…
argv1: .word 0x00
```



SP	0x7efff7e0	
0x7efff7e0	02	argc
0x7efff7e4	0x7efff8fd	argv[0]
0x7efff7e8	0x7efff92c	argv[1]
0x7efff92c	0x6c6c6568	‘l’ ‘l’ ‘e’ ‘h’
0x7efff7f0	0x78742e6f	‘x’ ‘t’ ‘.’ ‘o’
0x7efff7f4	0x….0074	… 0 ‘t’

```
$ ./argv hello.txt
```

Octal

- POSIX uses octal for system call flags
- base 8
- `0`*dddd…*

decimal	hex	octal	decimal	hex	octal
0	0	0	8	8	10
1	1	1	9	9	11
2	2	2	10	A	12
3	3	3	11	B	13
4	4	4	12	C	14
5	5	5	13	D	15
6	6	6	14	E	16
7	7	7	15	F	17

File open

- open
  - R7 == 5
  - R0 == address of path string
  - R1 == flags - see `fcntl.h`
    - read only: O_RDONLY == 0000
    - write only: O_WRONLY == 0001
    - create new write file: O_CREAT == 0100
      - OR with O_WRONLY

File open

- open
  - R2 == mode
    - only required for O_CREAT
    - specifies access permissions
- returns R0 - file descriptor

File access permissions

```
[greg@amaterasu ~]$ ls -l
total 378056
-rwxr-x---   1 greg staff     19456 Mar 26  2003 address.doc
-rwxr-x---   1 greg staff   1375170 Sep 15  2003 addresses
-rw-rw-rw-   1 greg staff       145 Aug 28  2003 AdobeFnt.lst
-rw-r--r--   1 greg staff    525312 Nov 21  2012 archive.pst
...
```

- can control owner, group & world access
- 3 bits each:
  – r == read == 100
  – w == write == 010
  – x == executable == 001

File access permissions

- e.g. rwx r-x ---
- == 111 101 000
- == 0750
- change access permissions in shell:

```
$ chmod access file
```

- in ARM let’s use rw-r--r--
- == 110 100 100
- == 0644

File close

- close
  - R7 == 6
  - R0 == file descriptor

Example - file copy

- open input from argv[1]
- open output from argv[2]
- read from input
- while still input do
  - write to output
  - read from input
- close files

Example - file copy

```
.data
.equ O_RDONLY, 0000
.equ O_WRONLY, 0001
.equ O_CREAT, 0100
access: .word 0644
char: .word 0x00
fin: .word 0x00
fout: .word 0x00
…
```

```
.global _start
_start:
@ open input argv[1]
      LDR R0, [SP,#8]
      MOV R1, #O_RDONLY
      BL open
      LDR R1, =fin
      STR R0, [R1]
```

Example - file copy

```
@ open output argv[2]
      LDR R0, [SP,#12]
      MOV R1, #O_WRONLY
      ORR R1, #O_CREAT
      LDR R3, =access
      LDR R2, [R3]
      BL open
      LDR R1, =fout
      STR R0, [R1]
```

```
loop: LDR R1, =fin
      LDR R0, [R1]
      LDR R1, =char
      MOV R2, #1
      BL read
      CMP R0, #0
      BEQ endl
      LDR R1, =fout
      LDR R0, [R1]
      LDR R1, =char
      MOV R2, #1
      BL write
      B loop
```

Example - file copy

```
endl:
@ close files
      LDR R1, =fin
      LDR R0, [R1]
      BL close
      LDR R1, =fout
      LDR R0, [R1]
      BL close
```

```
$ ./fcopy hello.txt
hello2.txt
$ ls -l hello2.txt
-rw-r--r-- 1 greg staff 63
Jan 20 14:09 hello2.txt
```

Library calls

- to call functions in C libraries
- procedure call standard depends on architecture
- follow AAPCS
  - ARM Architecture Procedure Call Standard
- pass parameters 1-4 in R0-R3
- pass other parameters on stack

Library calls

- need to make assembly program look like C
- change `_start` to `main`

`.global` *function*

- for each library function we wish to call

Library calls

- compile with `as` as before
  - `as` will generate `_start` from `main`
- link with `gcc` instead of `ld`
- `gcc` automatically links to `libc`
  - C standard library
- NB don’t mix system calls & library calls
  - different call conventions

Example: Q & A

```
main()
{  int n;
   printf(“How many beans make 5?”);
   scanf(“%d”,&n);
   if(n==5)
    printf(“Well done!\n”);
   else
    printf(“%d beans do not make 5!”,n);
}
```

Example: Q & A

```
...
.global printf
.global scanf

.data
f1: .asciz "How many beans make 5? "
f2: .asciz "%d"
f3: .asciz "Well done!\n"
f4: .asciz "%d beans do not make 5!\n"
n: .word 0x00
```

Example: Q & A

```
.global main

main:
     LDR R0, =f1
     BL printf

     LDR R0, =f2
     LDR R1, =n
     BL scanf
```

```
     LDR R0, =n
     LDR R1, [R0]
     CMP R1, #5
     BEQ yes
no:  LDR R0, =f4
     B print
yes: LDR R0, =f3
print:
     BL printf
     B _exit
...
```

Example: Q & A

```
$ as -g -o qa.o qa.s
$ gcc -o qa qa.o
$ ./qa
How many beans make 5? 4
4 beans do not make 5!
$
```

Example: file copy

```
main(int argc,char ** argv)
{  FILE * fin, * fout;
   int ch;
   fin = fopen(argv[1],"r");
   fout = fopen(argv[2],"w");
   ch = getc(fin);
   while(ch!=EOF)
   {  putc(ch,fout);
      ch = getc(fin);
   }
   fclose(fin);
   fclose(fout);
}
```

```
.global printf
.global .fopen
.global .fclose
.global getc
.global putc

.data
fin: .word 0x00
fout: .word 0x00
r: .asciz "r"
w: .asciz "w"
```

Example: file copy

```
main(int argc,char ** argv)
{  FILE * fin, * fout;
   int ch;
   fin = fopen(argv[1],"r");
   fout = fopen(argv[2],"w");
   ch = getc(fin);
   while(ch!=EOF)
   {  putc(ch,fout);
      ch = getc(fin);
   }
   fclose(fin);
   fclose(fout);
}
```

```
.global main
main:
@ fopen input argv[1]
       PUSH {R1}
       LDR R0, [R1,#0x04]
       LDR R1, =r
       BL fopen
       LDR R1, =fin
       STR R0, [R1]
```

- NB main is a function call
- `argc` in R0
- `argv` in R1

Example: file copy

```
main(int argc,char ** argv)
{   FILE * fin, * fout;
    int ch;
    fin = fopen(argv[1],"r");
    fout = fopen(argv[2],"w");
    ch = getc(fin);
    while(ch!=EOF)
    {   putc(ch,fout);
        ch = getc(fin);
    }
    fclose(fin);
    fclose(fout);
}
```

```
@ fopen output argv[2]
        POP {R1}
        LDR R0, [R1,#0x08]
        LDR R1, =w
        BL fopen
        LDR R1, =fout
        STR R0, [R1]
```

Example: file copy

```
main(int argc,char ** argv)
{  FILE * fin, * fout;
   int ch;
   fin = fopen(argv[1],"r");
   fout = fopen(argv[2],"w");
   ch = getc(fin);
   while(ch!=EOF)
   {  putc(ch,fout);
      ch = getc(fin);
   }
   fclose(fin);
   fclose(fout);
}
```

```
loop:  LDR R1, =fin
       LDR R0, [R1]
       BL getc
       CMP R0, #-1
       BEQ endl
       LDR R2, =fout
       LDR R1, [R2]
       BL putc
       B loop
endl:
```

Example: file copy

```
main(int argc,char ** argv)
{  FILE * fin, * fout;
   int ch;
   fin = fopen(argv[1],"r");
   fout = fopen(argv[2],"w");
   ch = getc(fin);
   while(ch!=EOF)
   {  putc(ch,fout);
      ch = getc(fin);
   }
   fclose(fin);
   fclose(fout);
}
```

```
@ fclose files
        LDR R1, =fin
        LDR R0, [R1]
        BL fclose
        LDR R1, =fout
        LDR R0, [R1]
        BL fclose

_exit:  MOV R7, #1
        MOV R0, #0
        SWI 0
```