Microprocessor & Computer Architecture (µpCA)

UE19CS252

Dr. D. C. Kiran

Department of
Computer Science and Engineering

Microprocessor & Computer Architecture (μpCA)

INTERRUPTS

Dr. D. C. Kiran

Department of Computer Science and Engineering

Microprocessor & Computer Architecture (μpCA)

Syllabus

Unit 1: Basic Processor Architecture and Design

- ~~Microprocessor Overview~~
- ~~CISC VS RISC~~
- ~~Introduction to ARM Processor & Applications~~
- ~~ARM Architecture Overview~~
- ~~Different ARM processor Modes~~
- ~~Register Bank~~
- ~~ARM Program structure~~
- ~~ARM Instruction Format~~
- **ARM INSTRUCTION SET**

  ~~Data Processing Instructions~~

  ~~Flow Control Instructions~~

  ~~Data Transfer Instructions~~

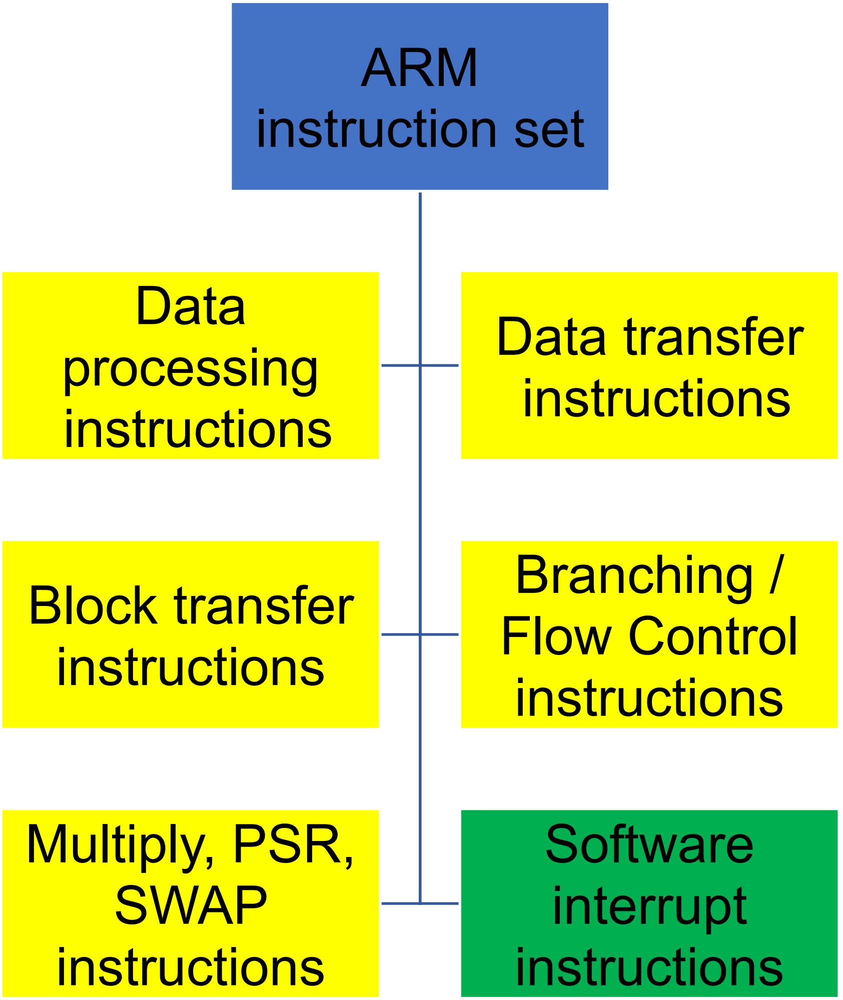
  ~~Block Transfer Instructions (Stack & Procedure Call)~~

  ~~Multiplication~~

  ~~MSR & MRS Instructions~~

  ~~Swap~~

  Interrupts



ARM instruction set

Data processing instructions

Data transfer instructions

Block transfer instructions

Branching / Flow Control instructions

Multiply, PSR, SWAP instructions

Software interrupt instructions

What is Interrupt/Exception?

- Main ()
- {
- :
- Doing something
- (e.g.browsing)
- :
- } ring

Can happen anytime
Depends on types of interrupts

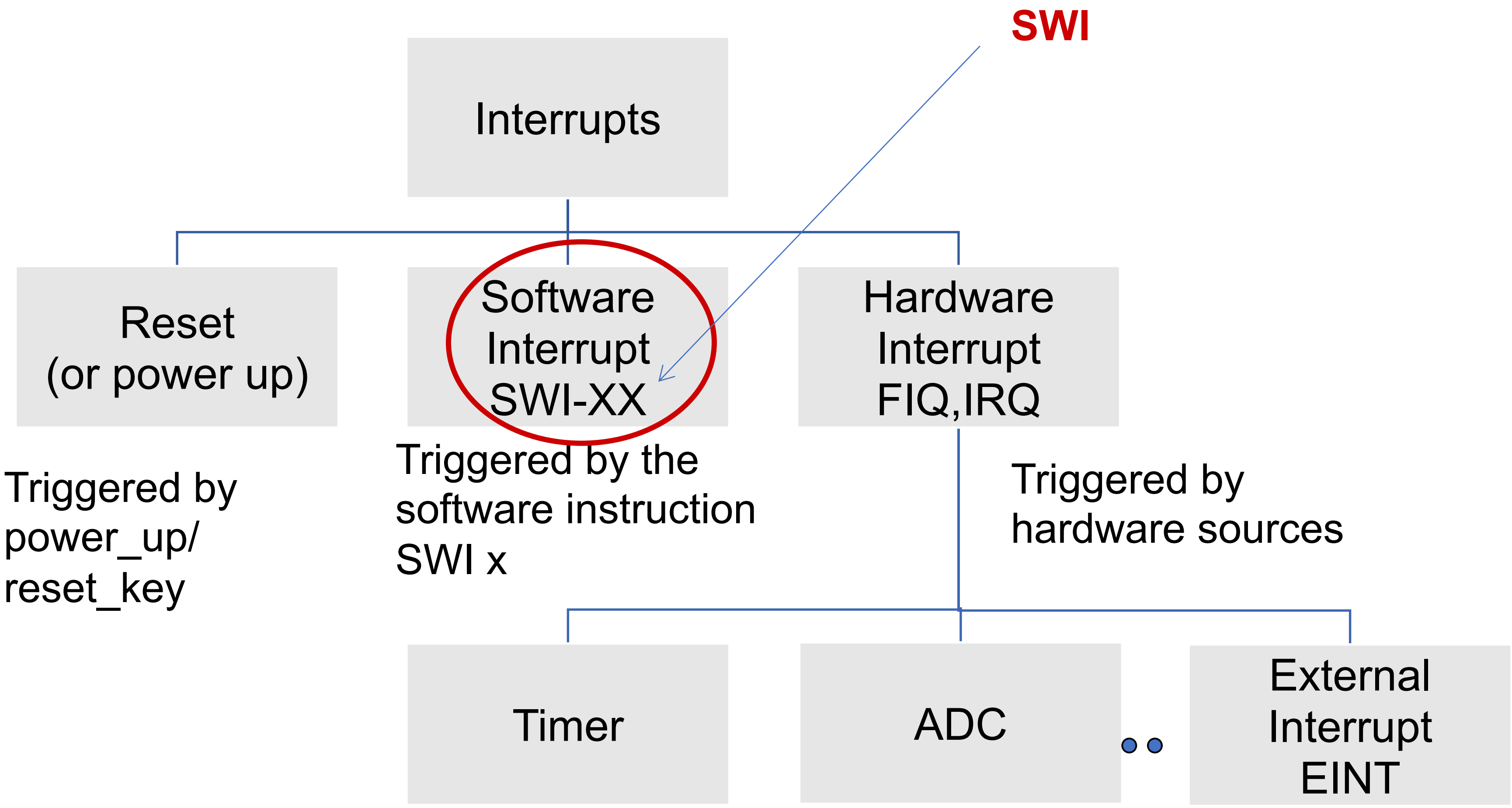
Phone rings

Phone rings

```
_isr() // Interrupt service routine
{
  some tasks (e.g. answer
                  telephone)

} //when finished,
  //goes back to main
```

Interrupts

-



SWI

Interrupts

Reset (or power up)

Software Interrupt SWI-XX

Hardware Interrupt FIQ,IRQ

Triggered by power_up/ reset_key

Triggered by the software instruction SWI x

Triggered by hardware sources

Timer

ADC

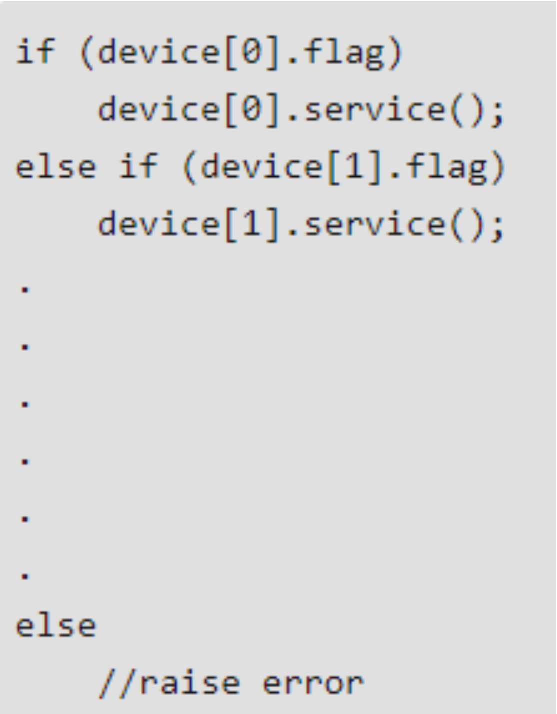
External Interrupt EINT

- Many sources of “events” during program execution
  - Application makes a system call
  - Software executes instruction illegally (e.g. divides by zero)
  - Peripheral needs attention or has completed a requested action
- How do we know that an event has occurred?
- Broadly, two options to “detect” events
  - Polling
    - We can repeatedly poll the app/processor/peripherals
    - When an event occurs, detect this via a poll and take action
  - Interrupts
    - Let the app/processors/peripheral notify us instead
    - Take action when such a notification occurs (or shortly later)

Reference

SOFTWARE METHOD – POLLING

- To serve multiple interrupts generated by devices simultaneously
- It’s a way to decide to which interrupt will be served first on priority.
- A service program will decide which interrupt to serve based on priority
- All the devices will be checked to see who has generated the interrupt.
- If Flag bit of a device is set, itz interrupt service is served.
- This process is slow

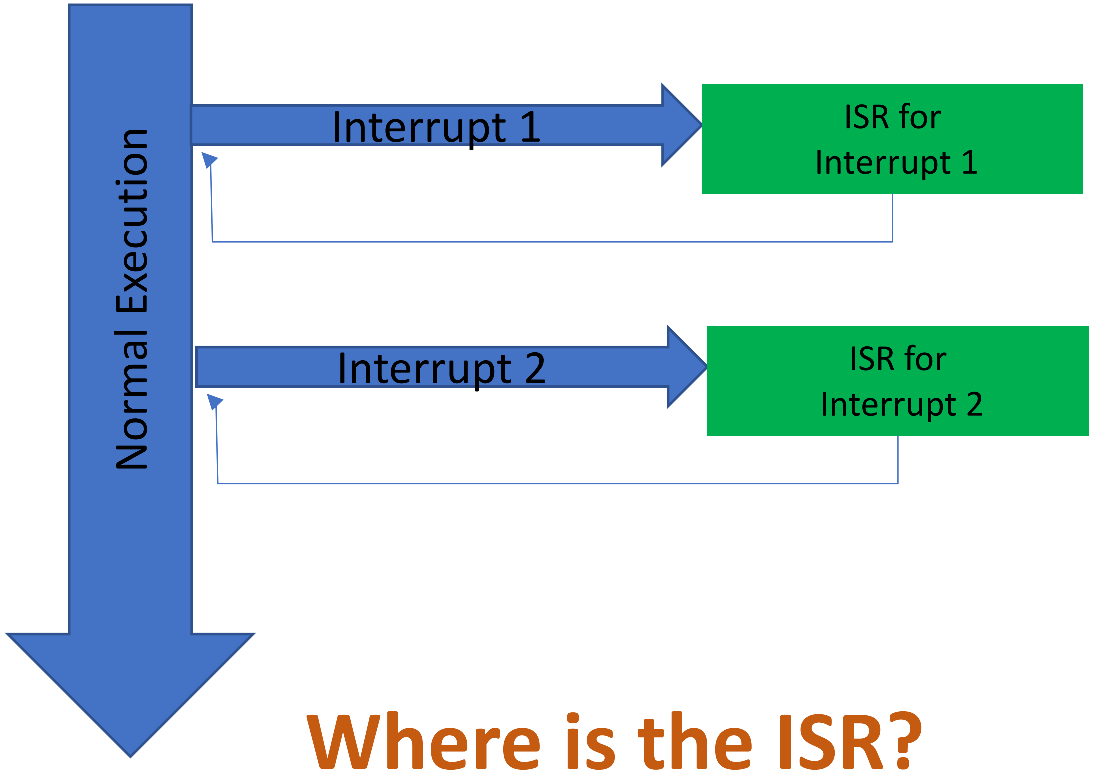


```
if (device[0].flag)
    device[0].service();
else if (device[1].flag)
    device[1].service();
.
.
.
.
.
.
else
    //raise error
```

Event Driven Tasks Execution

Each Event (Interrupt / Exception) has ISR

This is similar to the Sub-Routine call



Where is the ISR?

Somewhere in Code part of Main Memory

Interrupt Service Routine (ISR)

The ISR is a piece of code that's responsible for clearing the source of the interrupt.

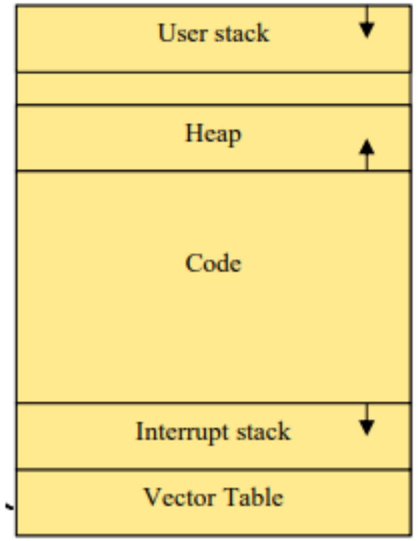
ISR is also called device driver in case of the devices and called exception or signal or trap handler in case of software interrupts

Event Driven Tasks Execution

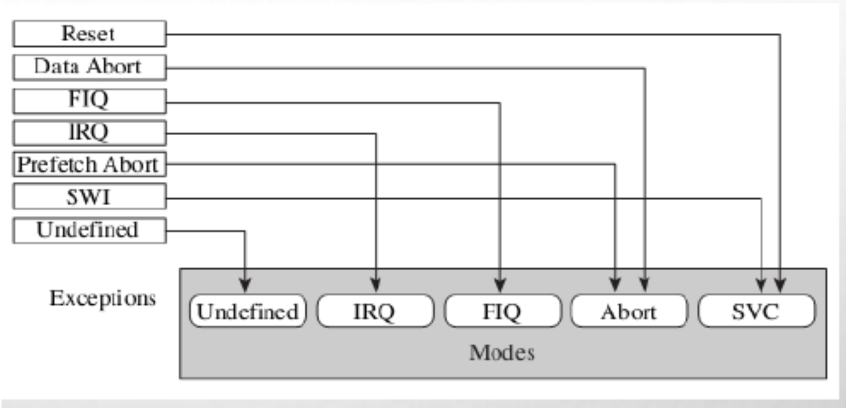
How the processor determines where the ISR is located in code memory for the specific interrupt?

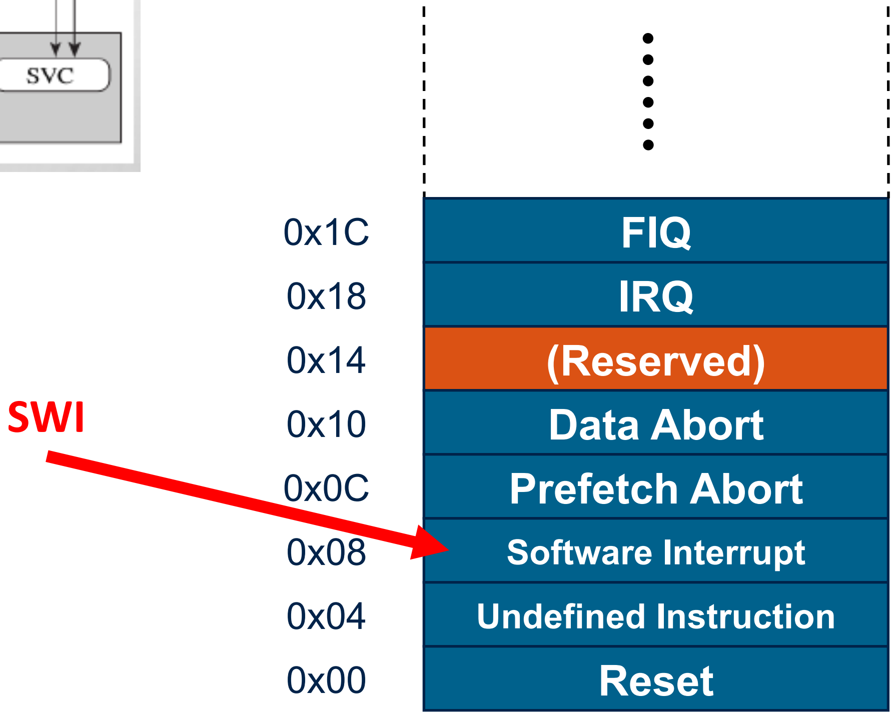
Microprocessor make use of Interrupt Vector Tables to find the starting address of ISR routines.

Main Memory



Reference





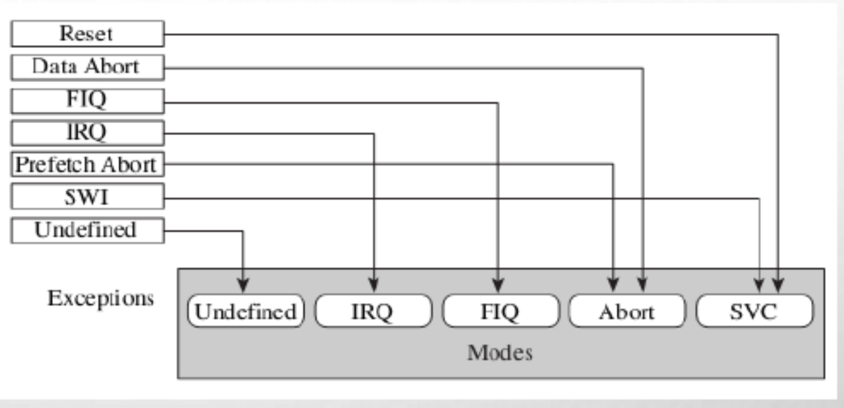
0x1C	FIQ
0x18	IRQ
0x14	(Reserved)
0x10	Data Abort
0x0C	Prefetch Abort
0x08	Software Interrupt
0x04	Undefined Instruction
0x00	Reset

Vector Table

Reference 1

Reference 2

Microprocessor & Computer Architecture (μpCA)



SWI

At this place in Memory, the Branch Instruction to the ISR can be found.

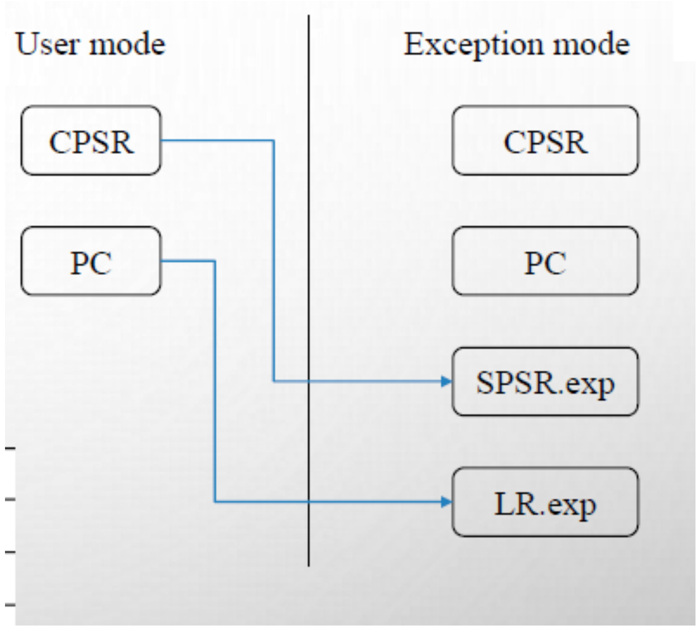
ldr pc, [pc, #_SWI_handler_offset]

	⋮
0x1C	FIQ
0x18	IRQ
0x14	(Reserved)
0x10	Data Abort
0x0C	Prefetch Abort
0x08	Software Interrupt
0x04	Undefined Instruction
0x00	Reset

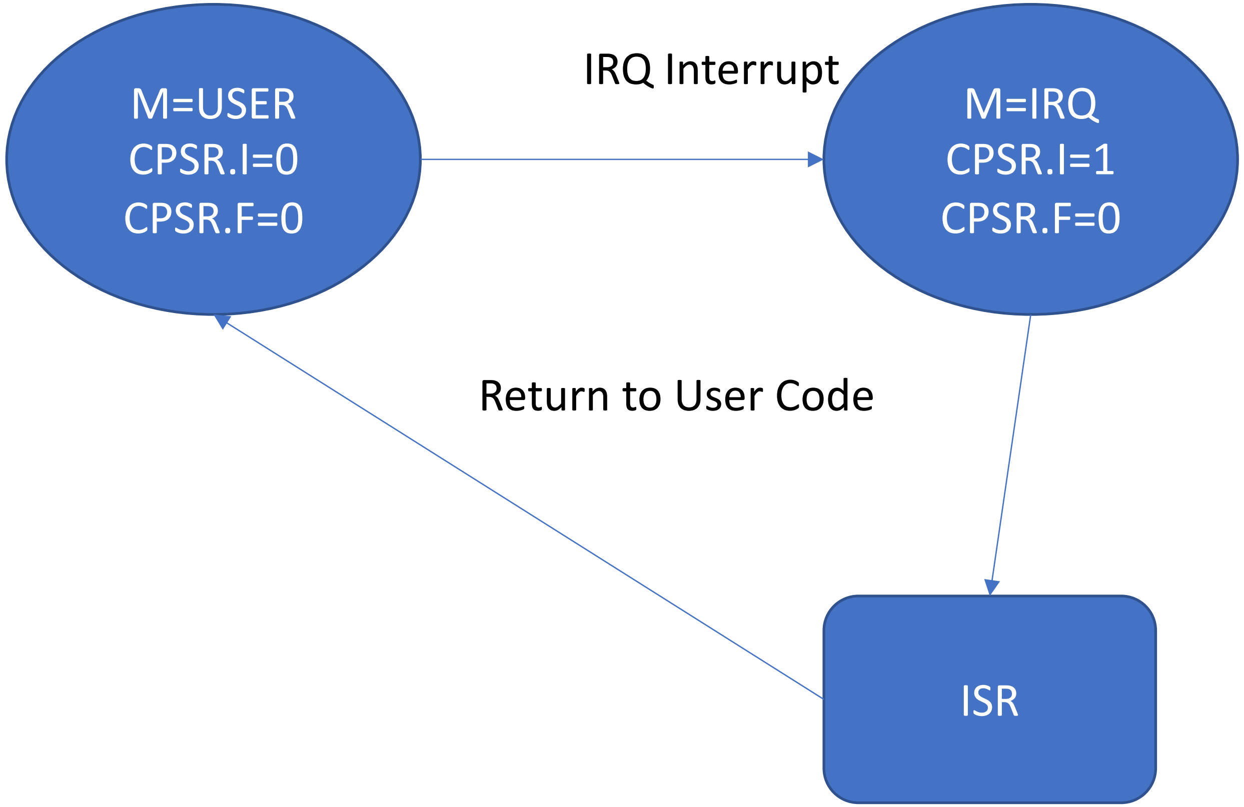
Vector Table

ARM Exception Handling

- Saves the CPSR to the SPSR of the Exception Mode
- Saves the PC to the LR of the Exception mode
- Sets the CPSR to the Exception Mode
- Sets PC to the address of the Exception Handler



Handling IRQ and FIQ Interrupts



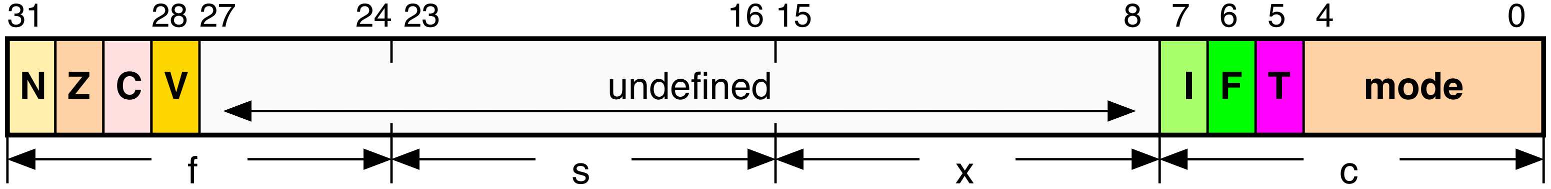
M=USER
CPSR.I=0
CPSR.F=0

IRQ Interrupt

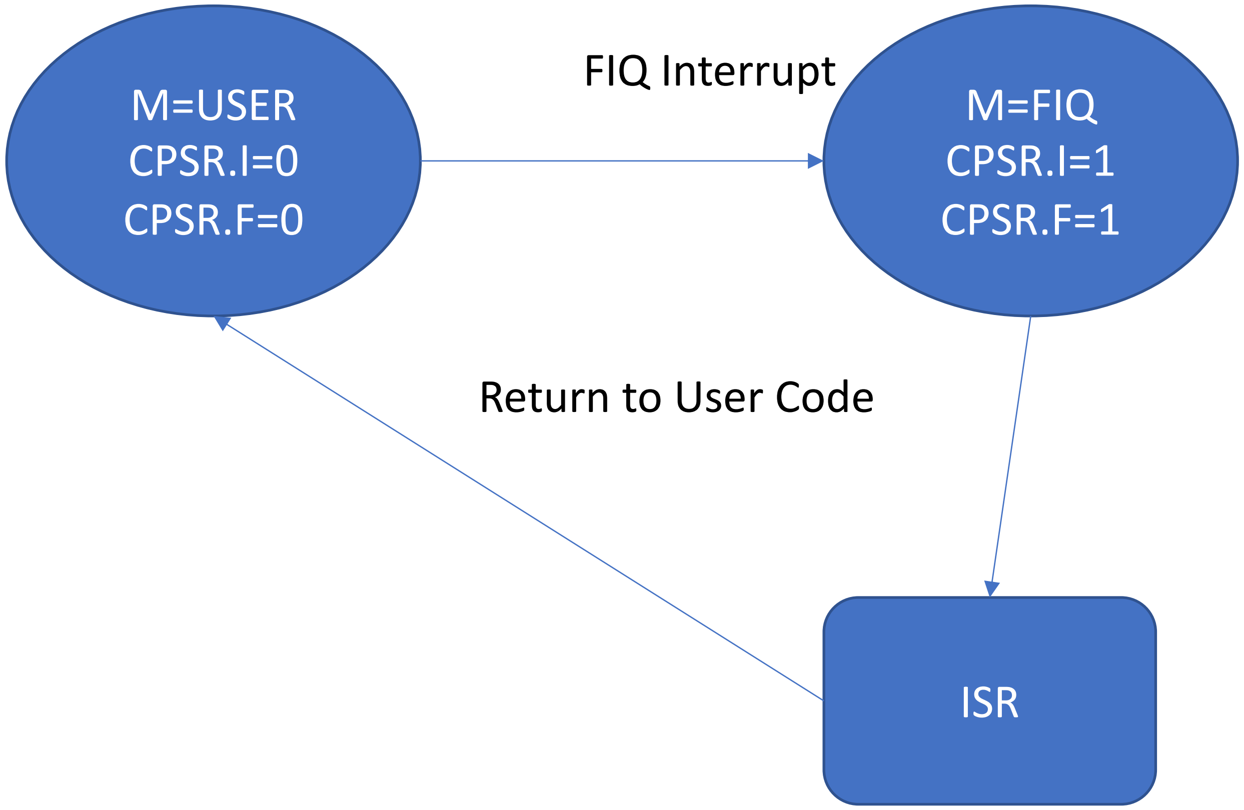
M=IRQ
CPSR.I=1
CPSR.F=0

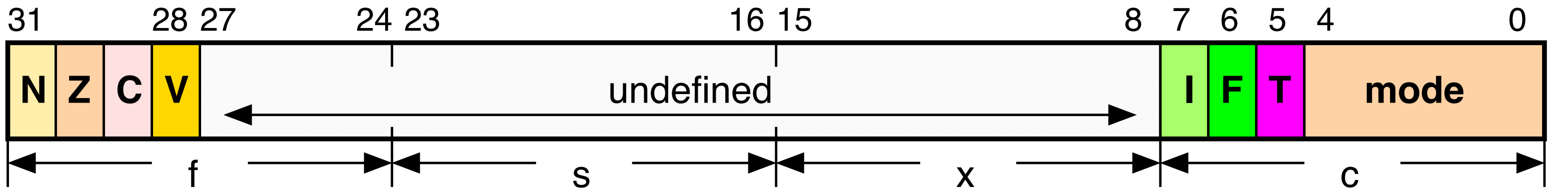
ISR

Return to User Code



Handling IRQ and FIQ Interrupts





Note the following

This is done by modifying the ***CPSR***, this is done using only 3 ARM instruction:

MRS	To read *CPSR*
MSR	To store in *CPSR*
BIC	Bit clear instruction
ORR	OR instruction

Enabling an IRQ/FIQ Interrupt:

```
MRS     r1, cpsr
BIC     r1, r1, #0x80/0x40
MSR     cpsr_c, r1
```

Disabling an IRQ/FIQ Interrupt:

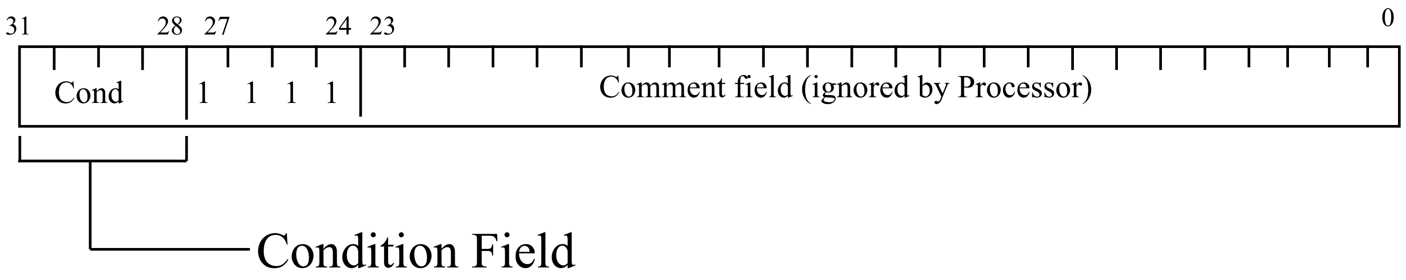
```
MRS     r1, cpsr
ORR     r1, r1, #0x80/0x40
MSR     cpsr_c, r1
```

I is 7th Bit in CPSR
F is 6th Bit in CPSR

0x80= 128 in binary 2^7
0x40 = 64 in binary 2^6

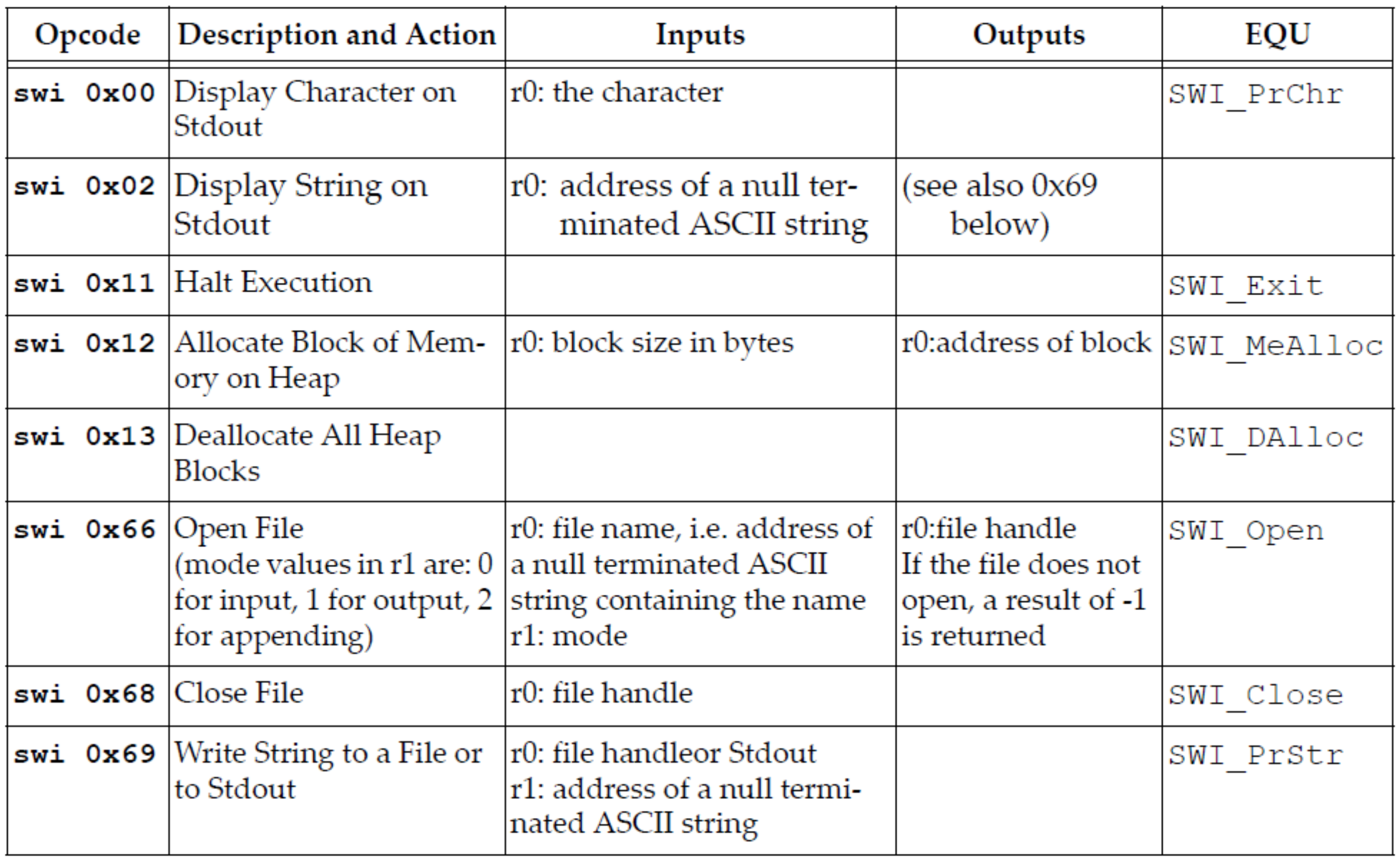
Microprocessor & Computer Architecture (μpCA)

SWI



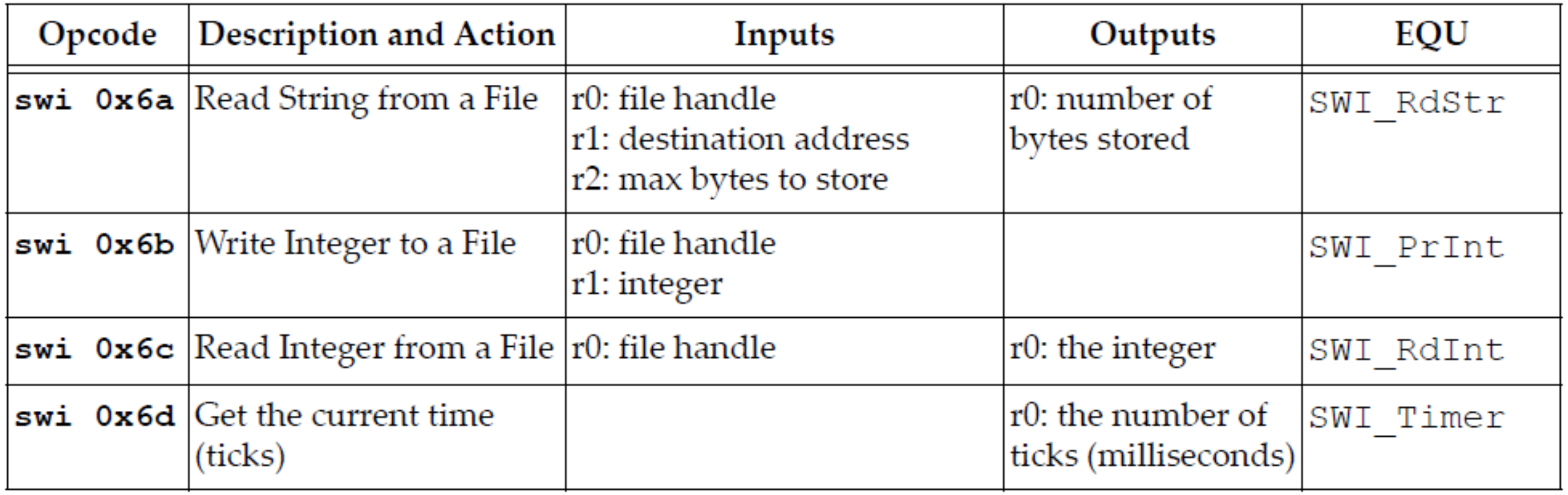
- In effect, a SWI is a user-defined instruction.
- It causes an exception trap to the SWI hardware vector (thus causing a change to supervisor mode, plus the associated state saving), thus causing the SWI exception handler to be called.
- The handler can then examine the comment field of the instruction to decide what operation has been requested.
- By making use of the SWI mechansim, an operating system can implement a set of privileged operations which applications running in user mode can request.

SWI



Opcode	Description and Action	Inputs	Outputs	EQU
`swi 0x00`	Display Character on Stdout	r0: the character		`SWI_PrChr`
`swi 0x02`	Display String on Stdout	r0: address of a null terminated ASCII string	(see also 0x69 below)	
`swi 0x11`	Halt Execution			`SWI_Exit`
`swi 0x12`	Allocate Block of Memory on Heap	r0: block size in bytes	r0:address of block	`SWI_MeAlloc`
`swi 0x13`	Deallocate All Heap Blocks			`SWI_DAlloc`
`swi 0x66`	Open File (mode values in r1 are: 0 for input, 1 for output, 2 for appending)	r0: file name, i.e. address of a null terminated ASCII string containing the name r1: mode	r0:file handle If the file does not open, a result of -1 is returned	`SWI_Open`
`swi 0x68`	Close File	r0: file handle		`SWI_Close`
`swi 0x69`	Write String to a File or to Stdout	r0: file handleor Stdout r1: address of a null terminated ASCII string		`SWI_PrStr`

SWI



Opcode	Description and Action	Inputs	Outputs	EQU
`swi 0x6a`	Read String from a File	r0: file handle r1: destination address r2: max bytes to store	r0: number of bytes stored	`SWI_RdStr`
`swi 0x6b`	Write Integer to a File	r0: file handle r1: integer		`SWI_PrInt`
`swi 0x6c`	Read Integer from a File	r0: file handle	r0: the integer	`SWI_RdInt`
`swi 0x6d`	Get the current time (ticks)		r0: the number of ticks (milliseconds)	`SWI_Timer`

A procedure to compute the statement in high level language using ARM ALP.

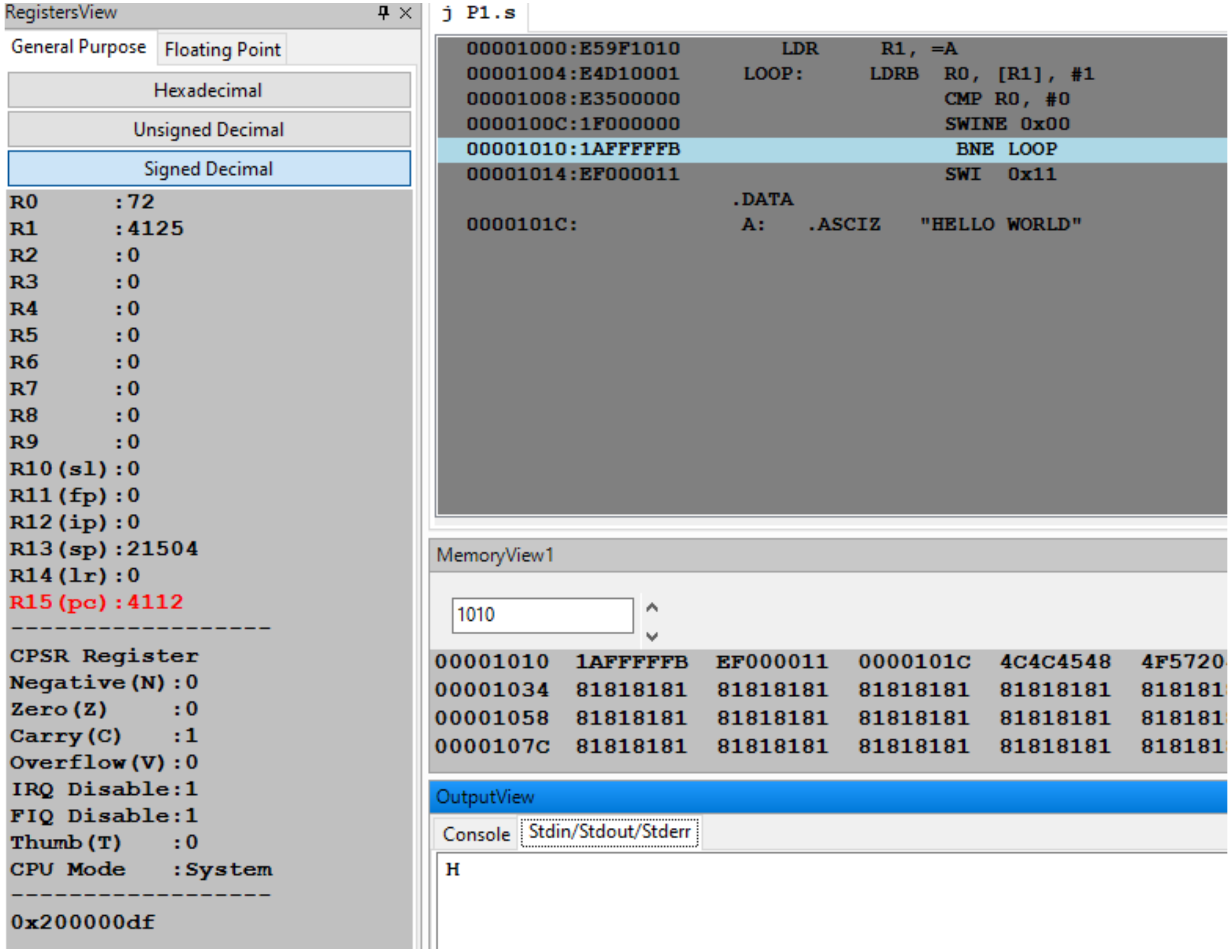
```
    if (R0==R1) R2++;

    MOV R0, #10
    MOV R1, #10
    BL GREAT
     SWI 0x11          ; terminate the program / logical end.

GREAT:  CMP   R0, R1
        ADDEQ R2, R2, #1
        MOV PC, LR
```

A program to display a string on the screen using ARM ALP.

printf(" Hello World");

```
          LDR    R1, =A
LOOP:     LDRB  R0, [R1], #1
          CMP R0, #0
          SWINE 0x00          ; display a character on the screen.
          BNE LOOP
          SWI  0x11           ; terminate the program.
.DATA
 A:   .ASCIZ   "HELLO WORLD"
```

Microprocessor & Computer Architecture (μpCA)

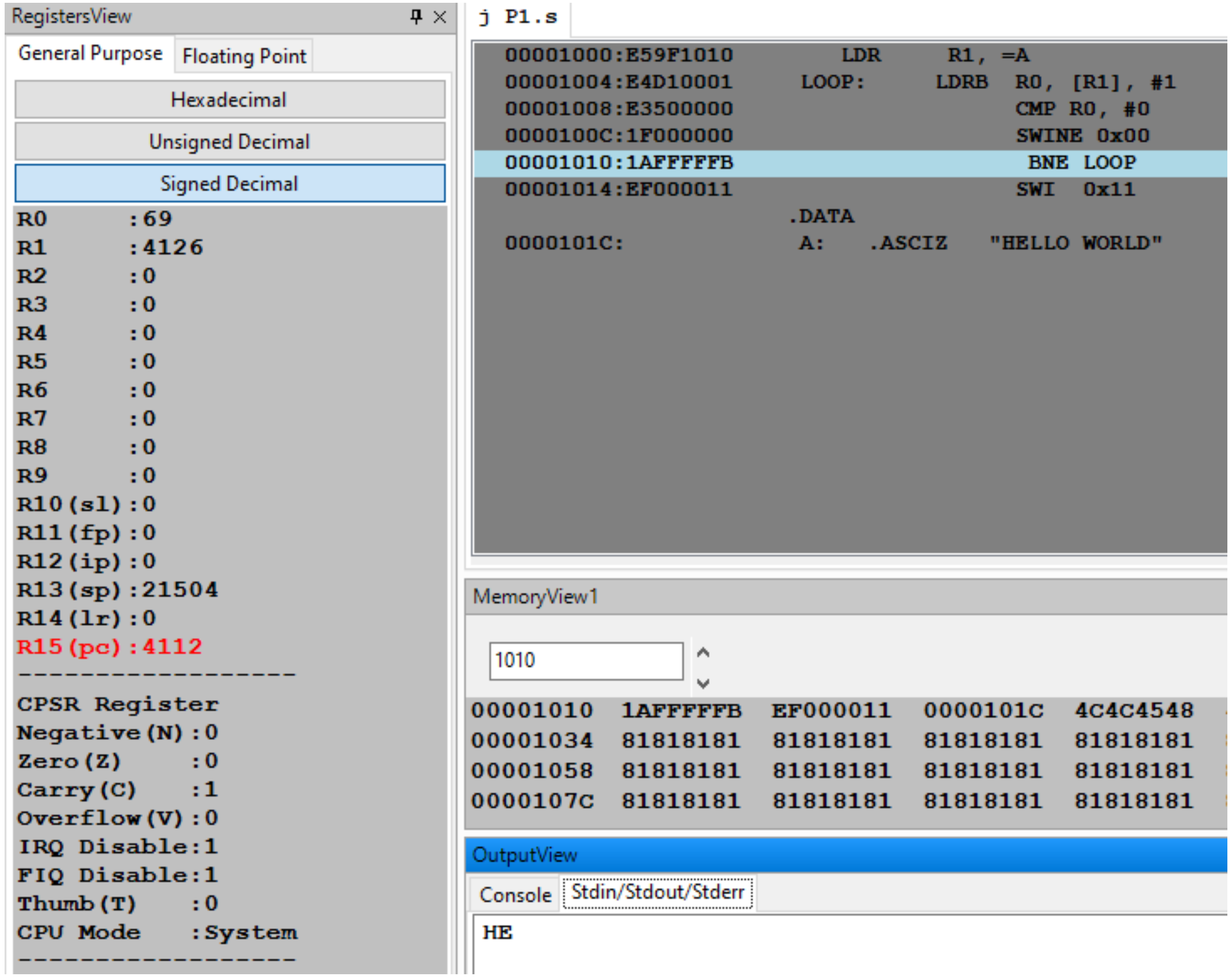


```
R0       :72
R1       :4125
R2       :0
R3       :0
R4       :0
R5       :0
R6       :0
R7       :0
R8       :0
R9       :0
R10(sl):0
R11(fp):0
R12(ip):0
R13(sp):21504
R14(lr):0
R15(pc):4112
------------------
CPSR Register
Negative(N):0
Zero(Z)    :0
Carry(C)   :1
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T)   :0
CPU Mode   :System
------------------
0x200000df
```

```
00001000:E59F1010         LDR      R1, =A
00001004:E4D10001     LOOP:      LDRB  R0, [R1], #1
00001008:E3500000                      CMP  R0, #0
0000100C:1F000000                      SWINE 0x00
00001010:1AFFFFFB                       BNE  LOOP
00001014:EF000011                      SWI   0x11
                            .DATA
0000101C:                    A:    .ASCIZ    "HELLO WORLD"
```

```
1010

00001010  1AFFFFFB  EF000011  0000101C  4C4C4548  4F5720
00001034  81818181  81818181  81818181  81818181  818181
00001058  81818181  81818181  81818181  81818181  818181
0000107C  81818181  81818181  81818181  81818181  818181
```

```
H
```



```
R0        :69
R1        :4126
R2        :0
R3        :0
R4        :0
R5        :0
R6        :0
R7        :0
R8        :0
R9        :0
R10(sl):0
R11(fp):0
R12(ip):0
R13(sp):21504
R14(lr):0
R15(pc):4112
------------------
CPSR Register
Negative(N):0
Zero(Z)    :0
Carry(C)   :1
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T)   :0
CPU Mode   :System
------------------
```

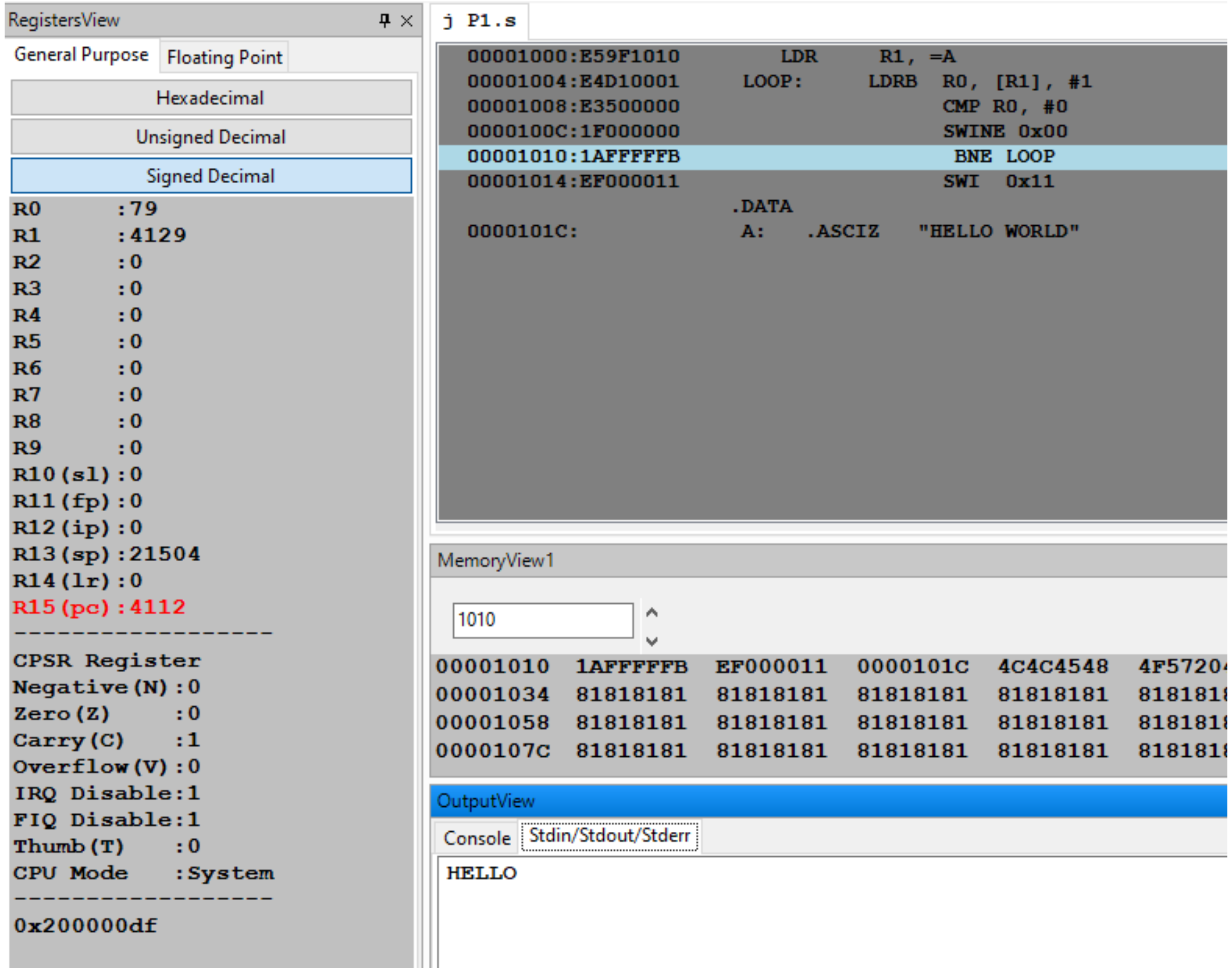
```
00001000:E59F1010          LDR       R1, =A
00001004:E4D10001      LOOP:      LDRB  R0, [R1], #1
00001008:E3500000                       CMP R0, #0
0000100C:1F000000                       SWINE 0x00
00001010:1AFFFFFB                        BNE LOOP
00001014:EF000011                       SWI  0x11
                                 .DATA
0000101C:                         A:    .ASCIZ    "HELLO WORLD"
```

1010

```
00001010  1AFFFFFB  EF000011  0000101C  4C4C4548
00001034  81818181  81818181  81818181  81818181
00001058  81818181  81818181  81818181  81818181
0000107C  81818181  81818181  81818181  81818181
```

```
HE
```

Microprocessor & Computer Architecture (μpCA)



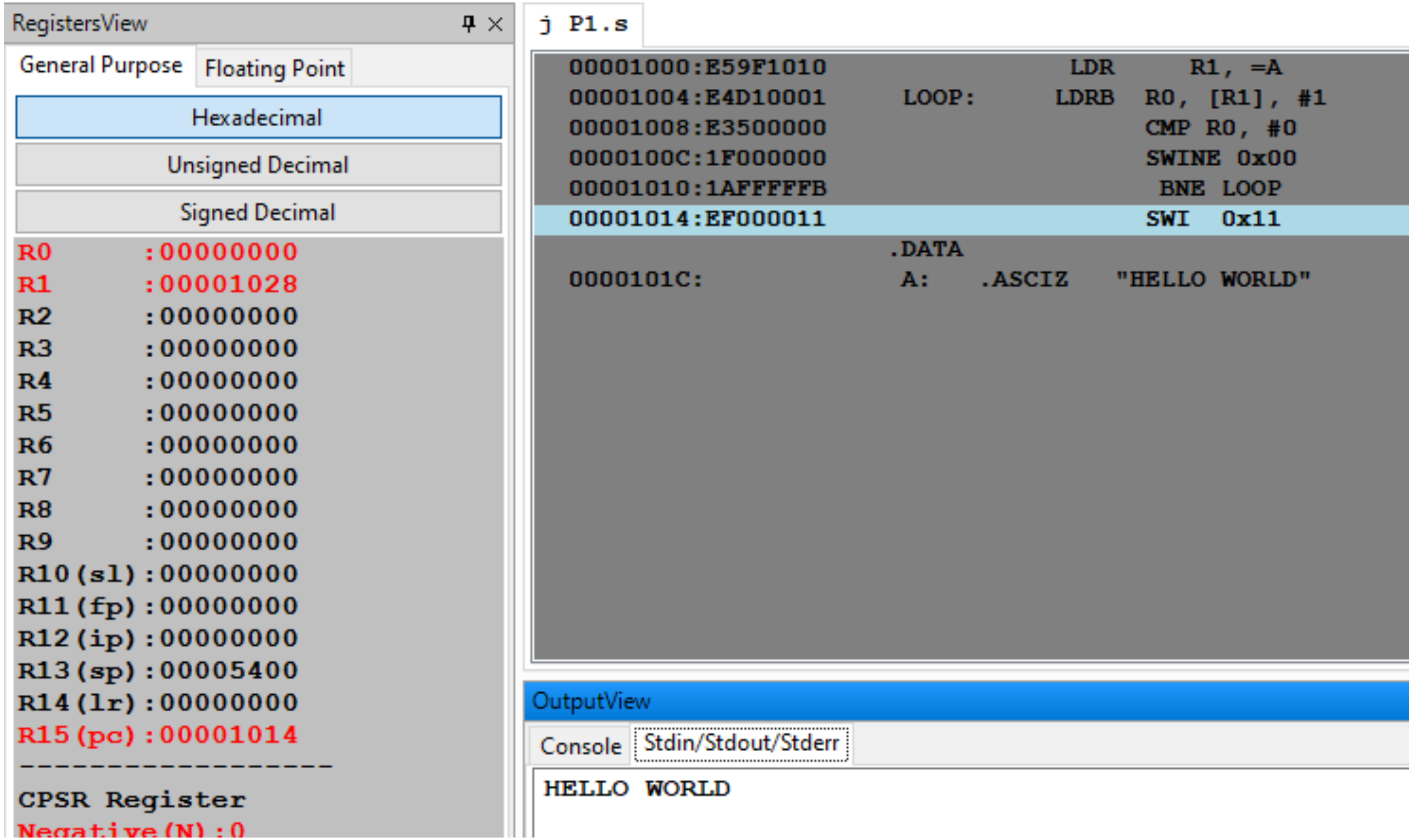
```
R0       :79
R1       :4129
R2       :0
R3       :0
R4       :0
R5       :0
R6       :0
R7       :0
R8       :0
R9       :0
R10(sl):0
R11(fp):0
R12(ip):0
R13(sp):21504
R14(lr):0
R15(pc):4112
------------------
CPSR Register
Negative(N):0
Zero(Z)    :0
Carry(C)   :1
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T)   :0
CPU Mode   :System
------------------
0x200000df
```

```
00001000:E59F1010        LDR      R1, =A
00001004:E4D10001     LOOP:      LDRB  R0, [R1], #1
00001008:E3500000                      CMP R0, #0
0000100C:1F000000                      SWINE 0x00
00001010:1AFFFFFB                       BNE  LOOP
00001014:EF000011                      SWI   0x11
                            .DATA
0000101C:                    A:    .ASCIZ    "HELLO WORLD"
```

```
00001010  1AFFFFFB  EF000011  0000101C  4C4C4548  4F5720
00001034  81818181  81818181  81818181  81818181  818181
00001058  81818181  81818181  81818181  81818181  818181
0000107C  81818181  81818181  81818181  81818181  818181
```

```
HELLO
```

Microprocessor & Computer Architecture (µpCA)



```
R0       :00000000
R1       :00001028
R2       :00000000
R3       :00000000
R4       :00000000
R5       :00000000
R6       :00000000
R7       :00000000
R8       :00000000
R9       :00000000
R10(sl)  :00000000
R11(fp)  :00000000
R12(ip)  :00000000
R13(sp)  :00005400
R14(lr)  :00000000
R15(pc)  :00001014
-------------------
CPSR Register
Negative(N):0
```

```
00001000:E59F1010              LDR     R1, =A
00001004:E4D10001    LOOP:     LDRB  R0, [R1], #1
00001008:E3500000                    CMP R0, #0
0000100C:1F000000                    SWINE 0x00
00001010:1AFFFFFB                     BNE  LOOP
00001014:EF000011                    SWI   0x11
                            .DATA
0000101C:                    A:    .ASCIZ    "HELLO WORLD"
```

```
HELLO WORLD
```

A procedure to display a string on the screen using ARM ALP.

// printf (“ Hello World”);

```
            LDR    R0, =A
   SWI     0x02          ; display a string on the screen
   SWI     0x11
.DATA

 A:   .ASCIZ   “HELLO WORLD”
```

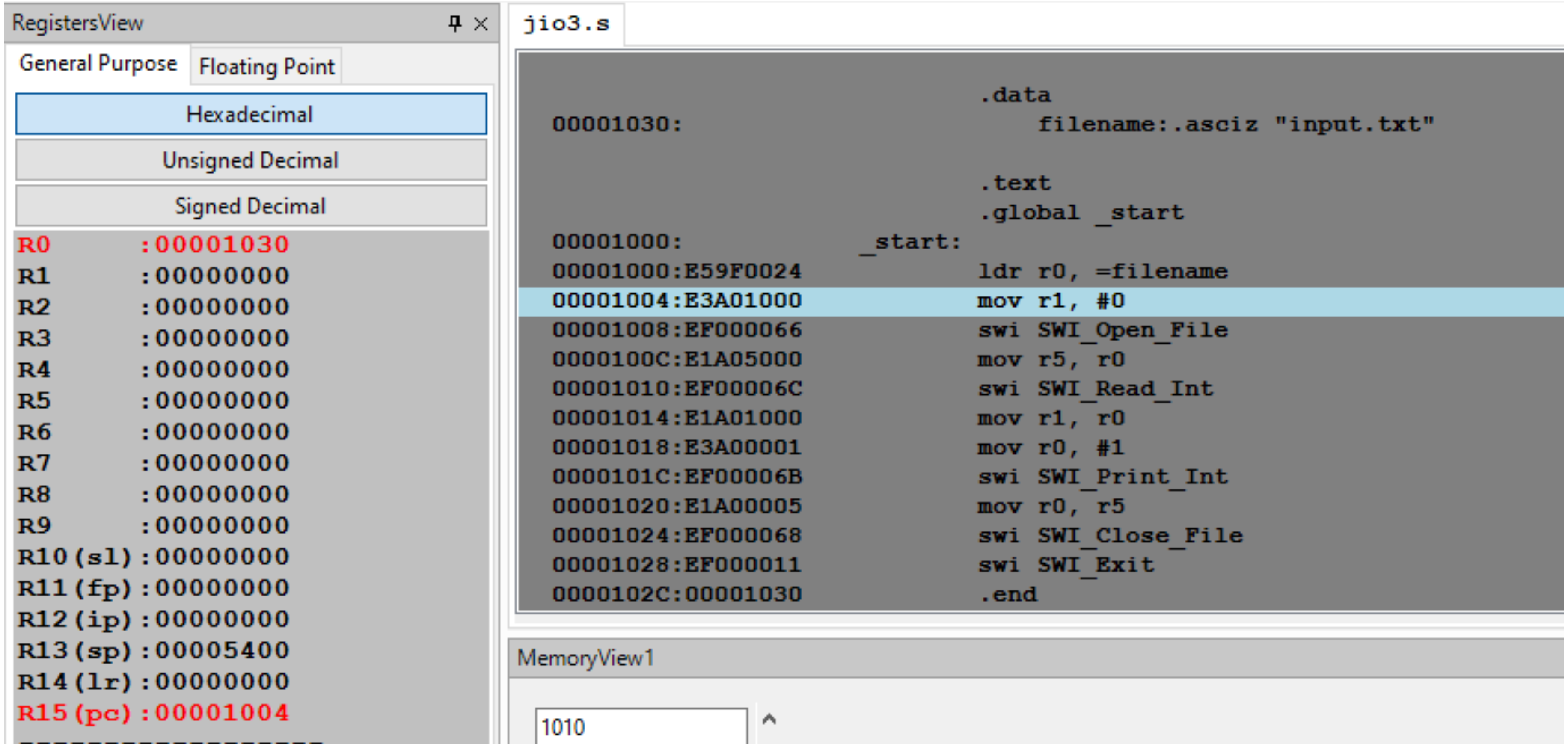
Program to Read an Integer from file and Display

```
.equ SWI_Open_File, 0x66
.equ SWI_Read_Int, 0x6C
.equ SWI_Print_Int, 0x6B
.equ SWI_Close_File, 0x68
.equ SWI_Exit, 0x11
```

```
        .data
    filename:.asciz "input.txt"

        .text
        .global _start
_start:
        ldr r0, =filename
        mov r1, #0
        swi SWI_Open_File
        mov r5, r0
        swi SWI_Read_Int
        mov r1, r0
        mov r0, #1
        swi SWI_Print_Int
        mov r0, r5
        swi SWI_Close_File
        swi SWI_Exit
        .end
```

Microprocessor & Computer Architecture (μpCA)

Program to Read an Integer from file and Display



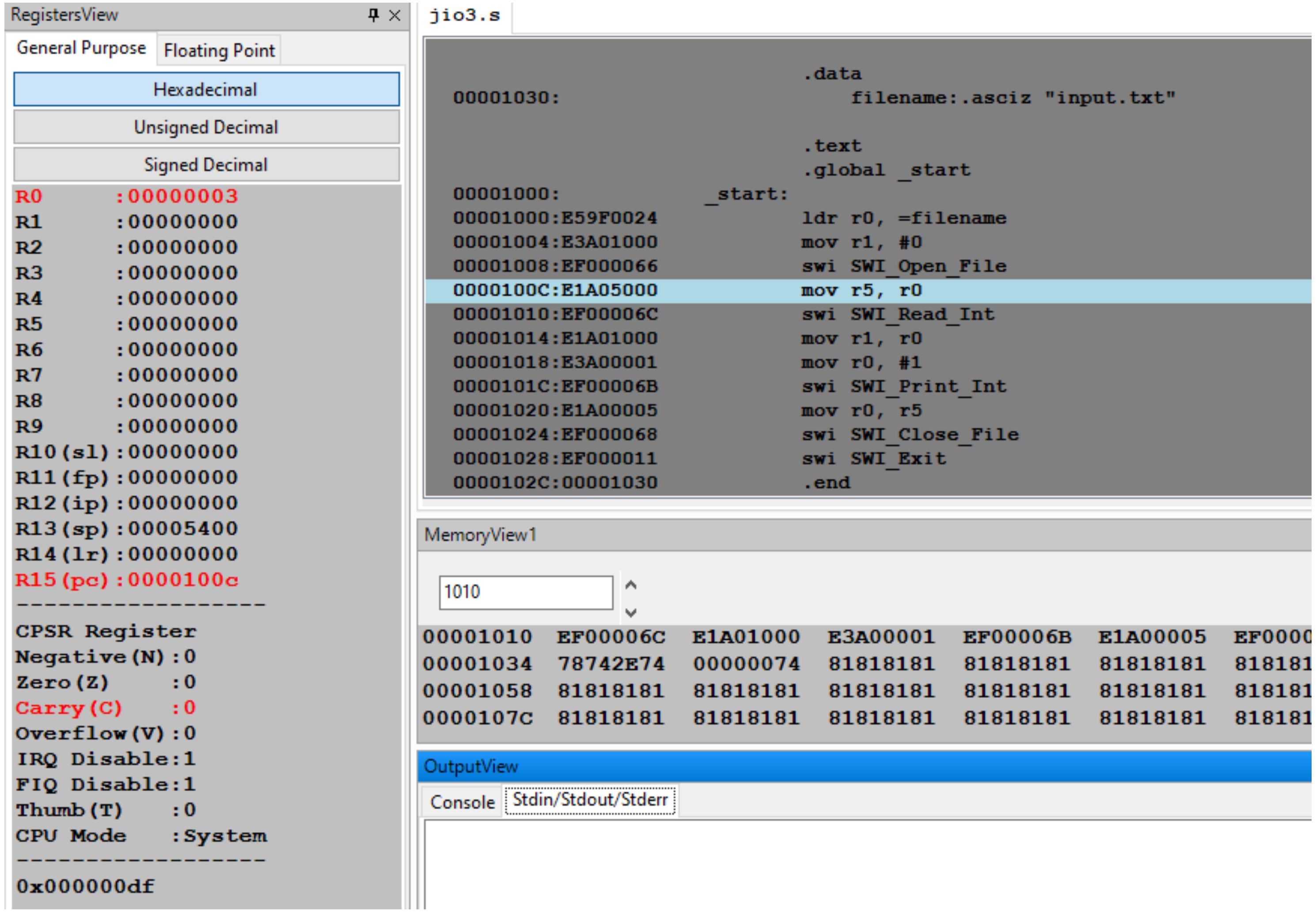
```
R0      :00001030
R1      :00000000
R2      :00000000
R3      :00000000
R4      :00000000
R5      :00000000
R6      :00000000
R7      :00000000
R8      :00000000
R9      :00000000
R10(sl) :00000000
R11(fp) :00000000
R12(ip) :00000000
R13(sp) :00005400
R14(lr) :00000000
R15(pc) :00001004
```

```
                                  .data
00001030:                             filename:.asciz "input.txt"

                                  .text
                                  .global _start
00001000:                 _start:
00001000:E59F0024                 ldr r0, =filename
00001004:E3A01000                 mov r1, #0
00001008:EF000066                 swi SWI_Open_File
0000100C:E1A05000                 mov r5, r0
00001010:EF00006C                 swi SWI_Read_Int
00001014:E1A01000                 mov r1, r0
00001018:E3A00001                 mov r0, #1
0000101C:EF00006B                 swi SWI_Print_Int
00001020:E1A00005                 mov r0, r5
00001024:EF000068                 swi SWI_Close_File
00001028:EF000011                 swi SWI_Exit
0000102C:00001030                 .end
```

1010

Microprocessor & Computer Architecture (μpCA)

Program to Read an Integer from file and Display



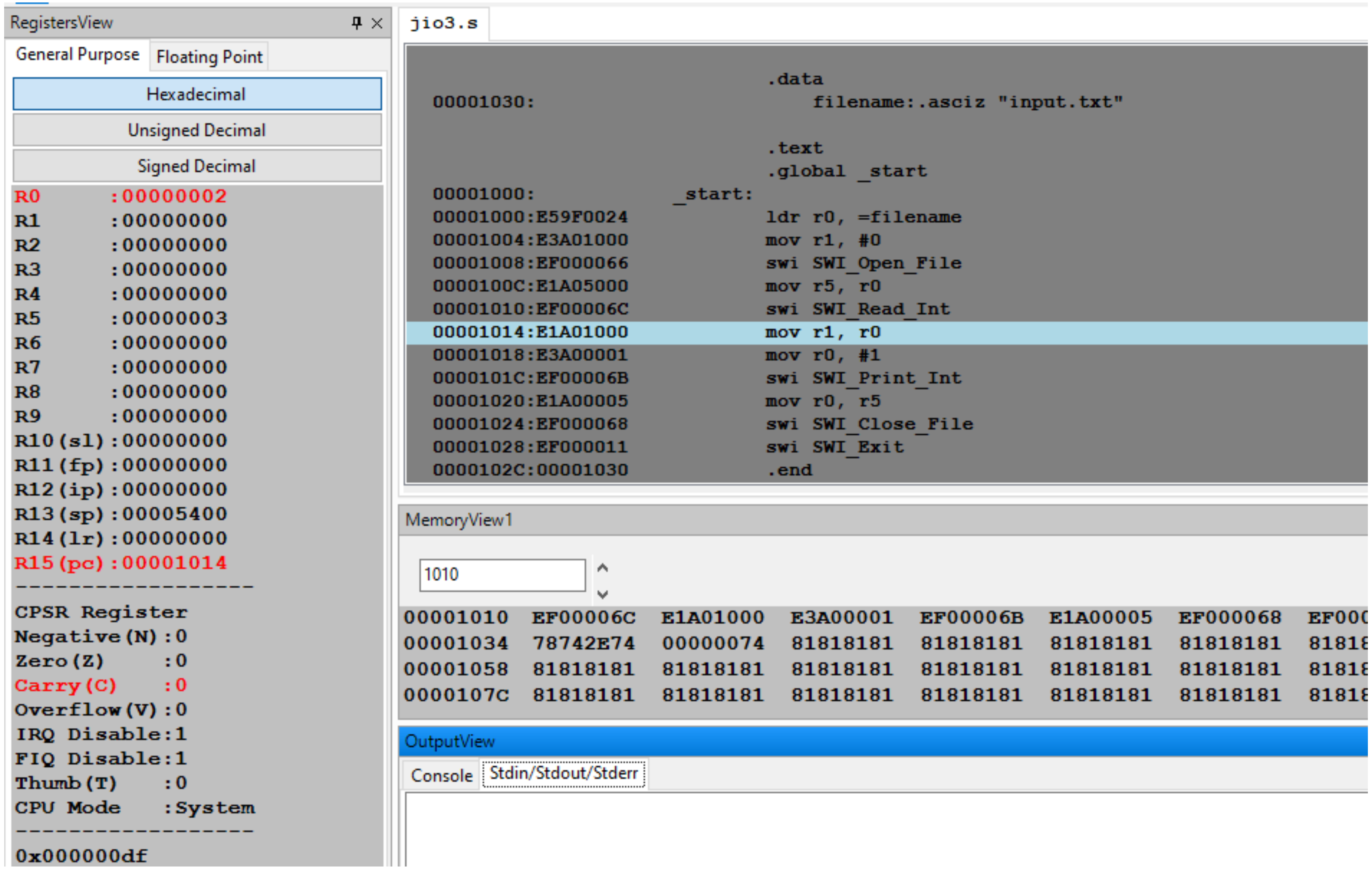
```
R0      :00000003
R1      :00000000
R2      :00000000
R3      :00000000
R4      :00000000
R5      :00000000
R6      :00000000
R7      :00000000
R8      :00000000
R9      :00000000
R10(sl) :00000000
R11(fp) :00000000
R12(ip) :00000000
R13(sp) :00005400
R14(lr) :00000000
R15(pc) :0000100c
------------------
CPSR Register
Negative(N):0
Zero(Z)    :0
Carry(C)   :0
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T)   :0
CPU Mode   :System
------------------
0x000000df
```

```
                                        .data
00001030:                                   filename:.asciz "input.txt"

                                        .text
                                        .global _start
00001000:                       _start:
00001000:E59F0024                       ldr r0, =filename
00001004:E3A01000                       mov r1, #0
00001008:EF000066                       swi SWI_Open_File
0000100C:E1A05000                       mov r5, r0
00001010:EF00006C                       swi SWI_Read_Int
00001014:E1A01000                       mov r1, r0
00001018:E3A00001                       mov r0, #1
0000101C:EF00006B                       swi SWI_Print_Int
00001020:E1A00005                       mov r0, r5
00001024:EF000068                       swi SWI_Close_File
00001028:EF000011                       swi SWI_Exit
0000102C:00001030                       .end
```

```
00001010  EF00006C  E1A01000  E3A00001  EF00006B  E1A00005  EF0000
00001034  78742E74  00000074  81818181  81818181  81818181  818181
00001058  81818181  81818181  81818181  81818181  81818181  818181
0000107C  81818181  81818181  81818181  81818181  81818181  818181
```

Microprocessor & Computer Architecture (μpCA)

Program to Read an Integer from file and Display



```
R0        :00000002
R1        :00000000
R2        :00000000
R3        :00000000
R4        :00000000
R5        :00000003
R6        :00000000
R7        :00000000
R8        :00000000
R9        :00000000
R10(sl):00000000
R11(fp):00000000
R12(ip):00000000
R13(sp):00005400
R14(lr):00000000
R15(pc):00001014
------------------
CPSR Register
Negative(N):0
Zero(Z)    :0
Carry(C)   :0
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T)   :0
CPU Mode   :System
------------------
0x000000df
```

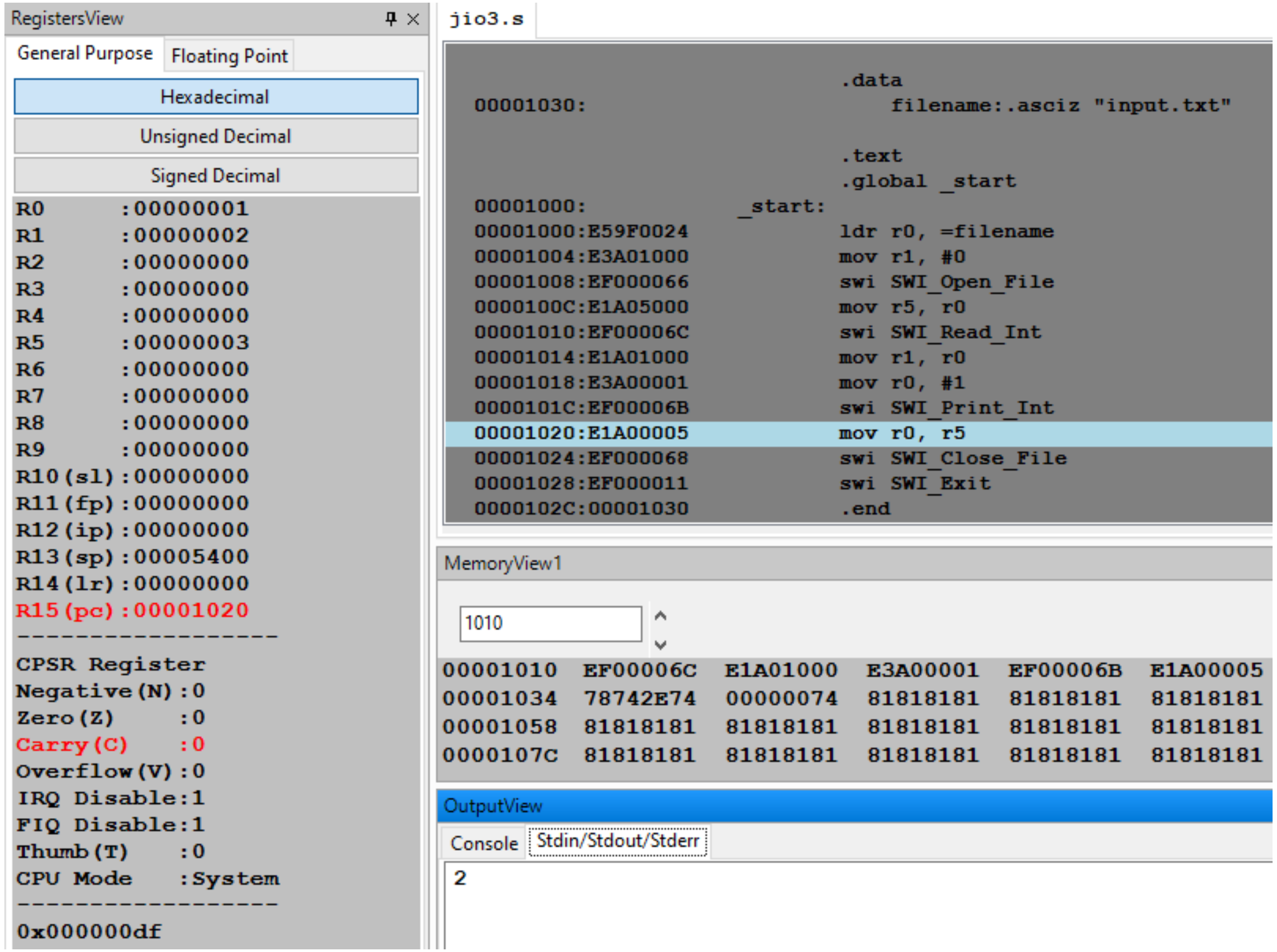
```
                              .data
00001030:                         filename:.asciz "input.txt"

                              .text
                              .global _start
00001000:               _start:
00001000:E59F0024             ldr r0, =filename
00001004:E3A01000             mov r1, #0
00001008:EF000066             swi SWI_Open_File
0000100C:E1A05000             mov r5, r0
00001010:EF00006C             swi SWI_Read_Int
00001014:E1A01000             mov r1, r0
00001018:E3A00001             mov r0, #1
0000101C:EF00006B             swi SWI_Print_Int
00001020:E1A00005             mov r0, r5
00001024:EF000068             swi SWI_Close_File
00001028:EF000011             swi SWI_Exit
0000102C:00001030             .end
```

1010

```
00001010  EF00006C  E1A01000  E3A00001  EF00006B  E1A00005  EF000068  EF000
00001034  78742E74  00000074  81818181  81818181  81818181  81818181  8181
00001058  81818181  81818181  81818181  81818181  81818181  81818181  8181
0000107c  81818181  81818181  81818181  81818181  81818181  81818181  8181
```

Microprocessor & Computer Architecture (μpCA)

Program to Read an Integer from file and Display



```
R0        :00000001
R1        :00000002
R2        :00000000
R3        :00000000
R4        :00000000
R5        :00000003
R6        :00000000
R7        :00000000
R8        :00000000
R9        :00000000
R10(sl):00000000
R11(fp):00000000
R12(ip):00000000
R13(sp):00005400
R14(lr):00000000
R15(pc):00001020
------------------
CPSR Register
Negative(N):0
Zero(Z)    :0
Carry(C)   :0
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T)   :0
CPU Mode   :System
------------------
0x000000df
```

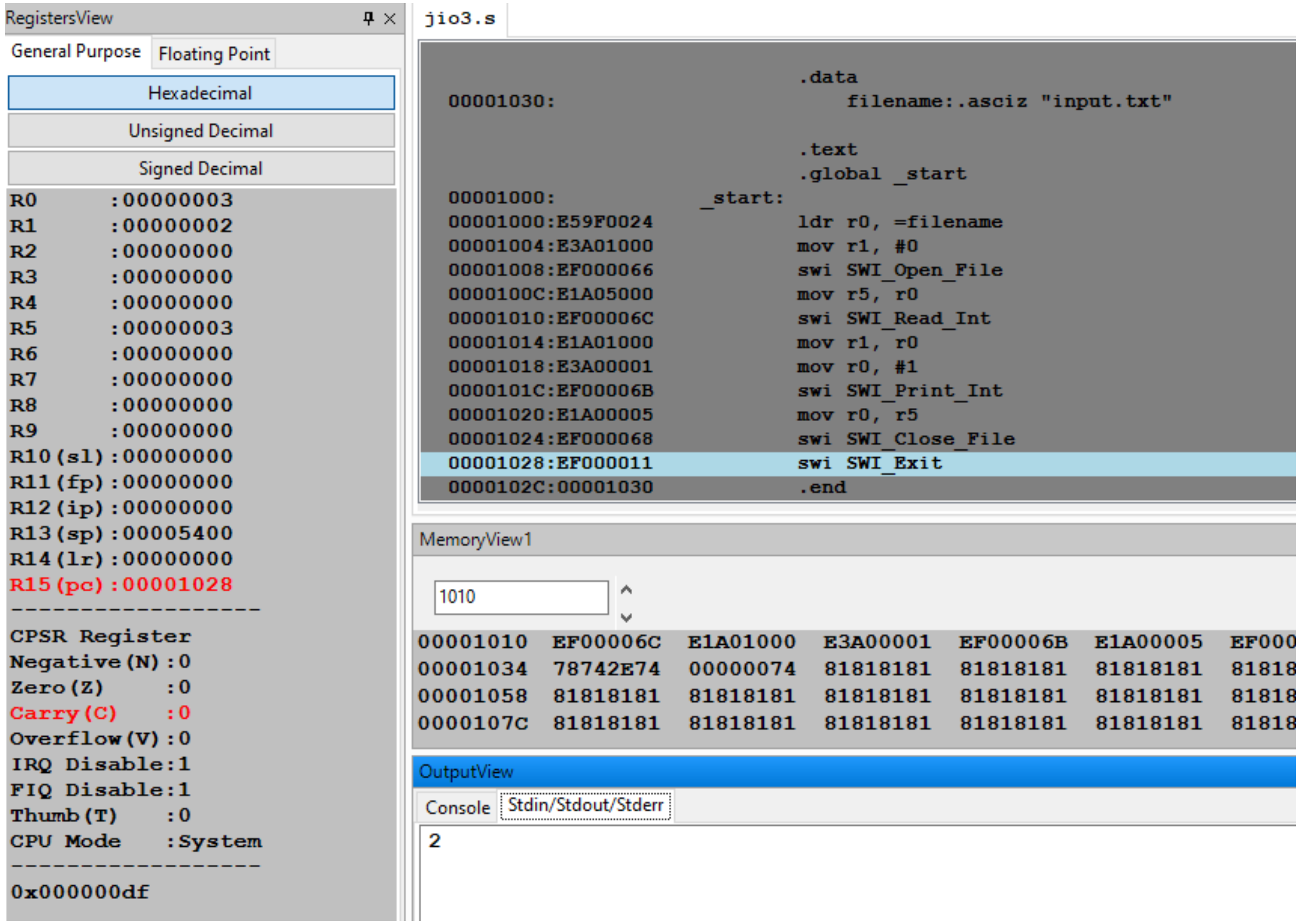
```
                                .data
00001030:                           filename:.asciz "input.txt"

                                .text
                                .global _start
00001000:                _start:
00001000:E59F0024               ldr r0, =filename
00001004:E3A01000               mov r1, #0
00001008:EF000066               swi SWI_Open_File
0000100C:E1A05000               mov r5, r0
00001010:EF00006C               swi SWI_Read_Int
00001014:E1A01000               mov r1, r0
00001018:E3A00001               mov r0, #1
0000101C:EF00006B               swi SWI_Print_Int
00001020:E1A00005               mov r0, r5
00001024:EF000068               swi SWI_Close_File
00001028:EF000011               swi SWI_Exit
0000102C:00001030               .end
```

```
1010

00001010  EF00006C  E1A01000  E3A00001  EF00006B  E1A00005
00001034  78742E74  00000074  81818181  81818181  81818181
00001058  81818181  81818181  81818181  81818181  81818181
0000107C  81818181  81818181  81818181  81818181  81818181
```

```
2
```

Microprocessor & Computer Architecture (μpCA)

Program to Read an Integer from file and Display



```
R0       :00000003
R1       :00000002
R2       :00000000
R3       :00000000
R4       :00000000
R5       :00000003
R6       :00000000
R7       :00000000
R8       :00000000
R9       :00000000
R10(sl):00000000
R11(fp):00000000
R12(ip):00000000
R13(sp):00005400
R14(lr):00000000
R15(pc):00001028
------------------
CPSR Register
Negative(N):0
Zero(Z)     :0
Carry(C)    :0
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T)    :0
CPU Mode    :System
------------------
0x000000df
```

```
                                .data
00001030:                           filename:.asciz "input.txt"

                                .text
                                .global _start
00001000:                  _start:
00001000:E59F0024               ldr r0, =filename
00001004:E3A01000               mov r1, #0
00001008:EF000066               swi SWI_Open_File
0000100C:E1A05000               mov r5, r0
00001010:EF00006C               swi SWI_Read_Int
00001014:E1A01000               mov r1, r0
00001018:E3A00001               mov r0, #1
0000101C:EF00006B               swi SWI_Print_Int
00001020:E1A00005               mov r0, r5
00001024:EF000068               swi SWI_Close_File
00001028:EF000011               swi SWI_Exit
0000102C:00001030               .end
```

1010

```
00001010  EF00006C  E1A01000  E3A00001  EF00006B  E1A00005  EF000
00001034  78742E74  00000074  81818181  81818181  81818181  81818
00001058  81818181  81818181  81818181  81818181  81818181  81818
0000107C  81818181  81818181  81818181  81818181  81818181  81818
```

```
2
```

Next Session

Instruction Encoding

PES UNIVERSITY ONLINE

THANK YOU

Dr. D. C. Kiran

Department of Computer Science and Engineering

dckiran@pes.edu

9829935135