Lab2：Debugging and Evaluation

Speaker: Chao-Chung Cheng

Directed by Prof. Tian-Sheuan Chang

March, 2004, NCTU

Goal of This Lab

- **Debug skills**
  - To debug both software running on processor and memory-mapped hardware design of the target platform.
- **Software cost estimation**
  - To estimate code sizes and benchmark performance
- **Profiling utility**
  - To estimate execution time of each function in an application
- **Memory configuration**
  - Performance/cost trade-off

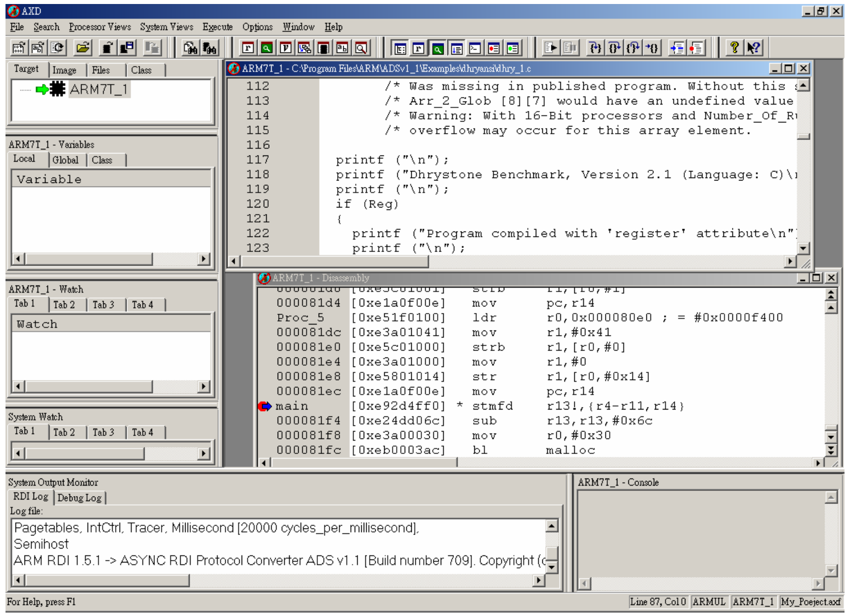
Outline

- *Debugging Skills*
- Software Quality Measurement (Evaluation)

Introduction

Debug software in ARM platform

- Graphic user interface (GUI) : ARM eXtended Debugger (**AXD**).
- Command line : ARM Symbolic Debugger (**armsd**).

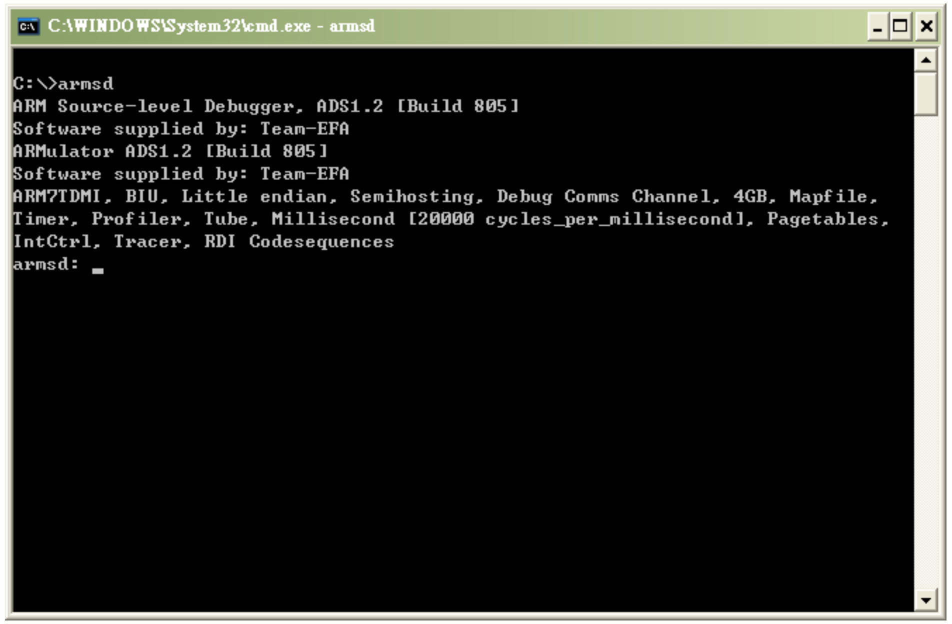


```
          /* Was missing in published program. Without this s
          /* Arr_2_Glob [8][7] would have an undefined value
          /* Warning: With 16-Bit processors and Number_Of_Ru
          /* overflow may occur for this array element.

  printf ("\n");
  printf ("Dhrystone Benchmark, Version 2.1 (Language: C)\n
  printf ("\n");
  if (Reg)
  {
    printf ("Program compiled with 'register' attribute\n")
    printf ("\n");
```

```
000081d4 [0xe1a0f00e]   mov    pc,r14
Proc_5   [0xe51f0100]   ldr    r0,0x000080e0 ; = #0x0000f400
000081dc [0xe3a01041]   mov    r1,#0x41
000081e0 [0xe5c01000]   strb   r1,[r0,#0]
000081e4 [0xe3a01000]   mov    r1,#0
000081e8 [0xe5801014]   str    r1,[r0,#0x14]
000081ec [0xe1a0f00e]   mov    pc,r14
main     [0xe92d4ff0] * stmfd  r13!,{r4-r11,r14}
000081f4 [0xe24dd06c]   sub    r13,r13,#0x6c
000081f8 [0xe3a00030]   mov    r0,#0x30
000081fc [0xeb0003ac]   bl     malloc
```

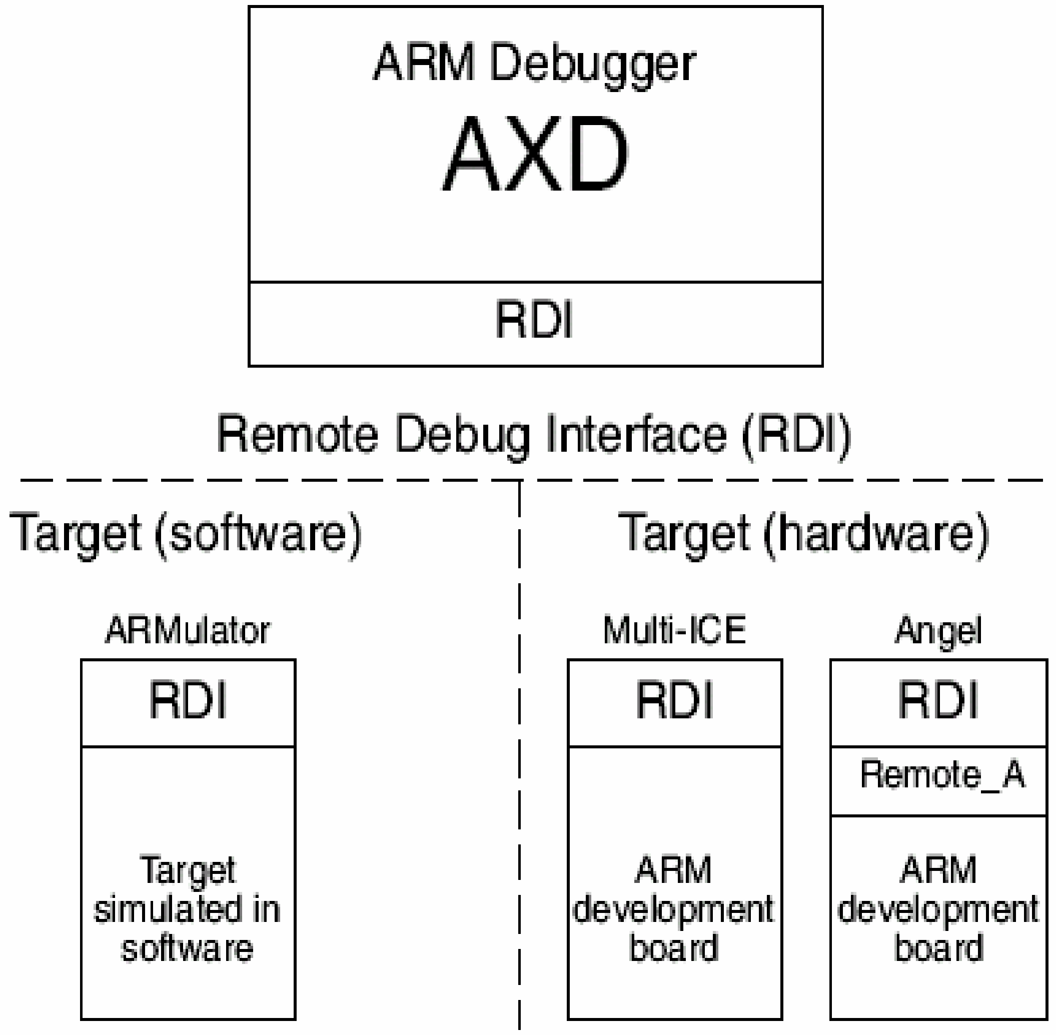
```
Pagetables, IntCtrl, Tracer, Millisecond [20000 cycles_per_millisecond],
Semihost
ARM RDI 1.5.1 -> ASYNC RDI Protocol Converter ADS v1.1 [Build number 709]. Copyright (c
```



```
C:\>armsd
ARM Source-level Debugger, ADS1.2 [Build 805]
Software supplied by: Team-EFA
ARMulator ADS1.2 [Build 805]
Software supplied by: Team-EFA
ARM7TDMI, BIU, Little endian, Semihosting, Debug Comms Channel, 4GB, Mapfile,
Timer, Profiler, Tube, Millisecond [20000 cycles_per_millisecond], Pagetables,
IntCtrl, Tracer, RDI Codesequences
armsd: _
```

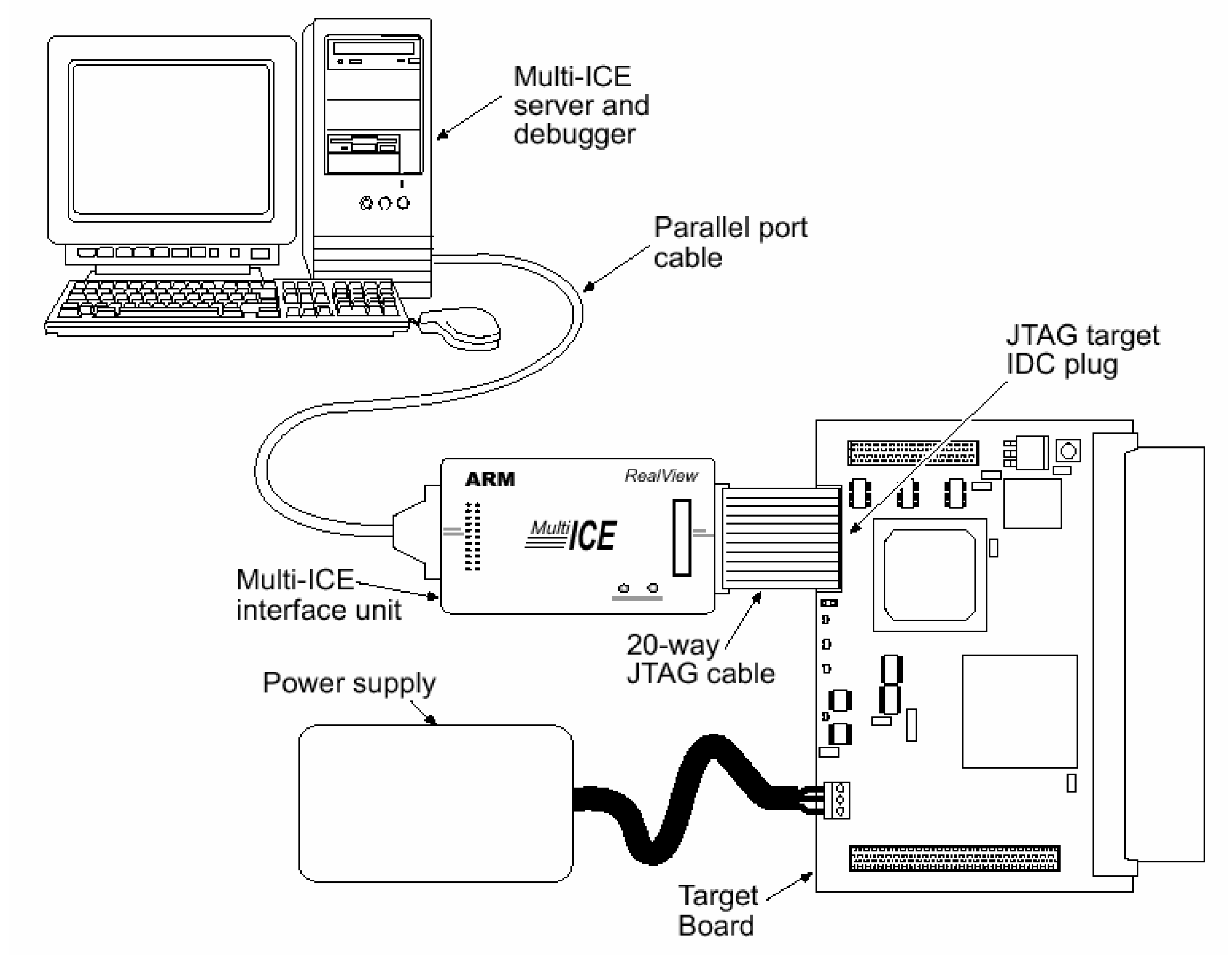
AXD armsd

Debug Target (1/2)



Debug Target (2/2)

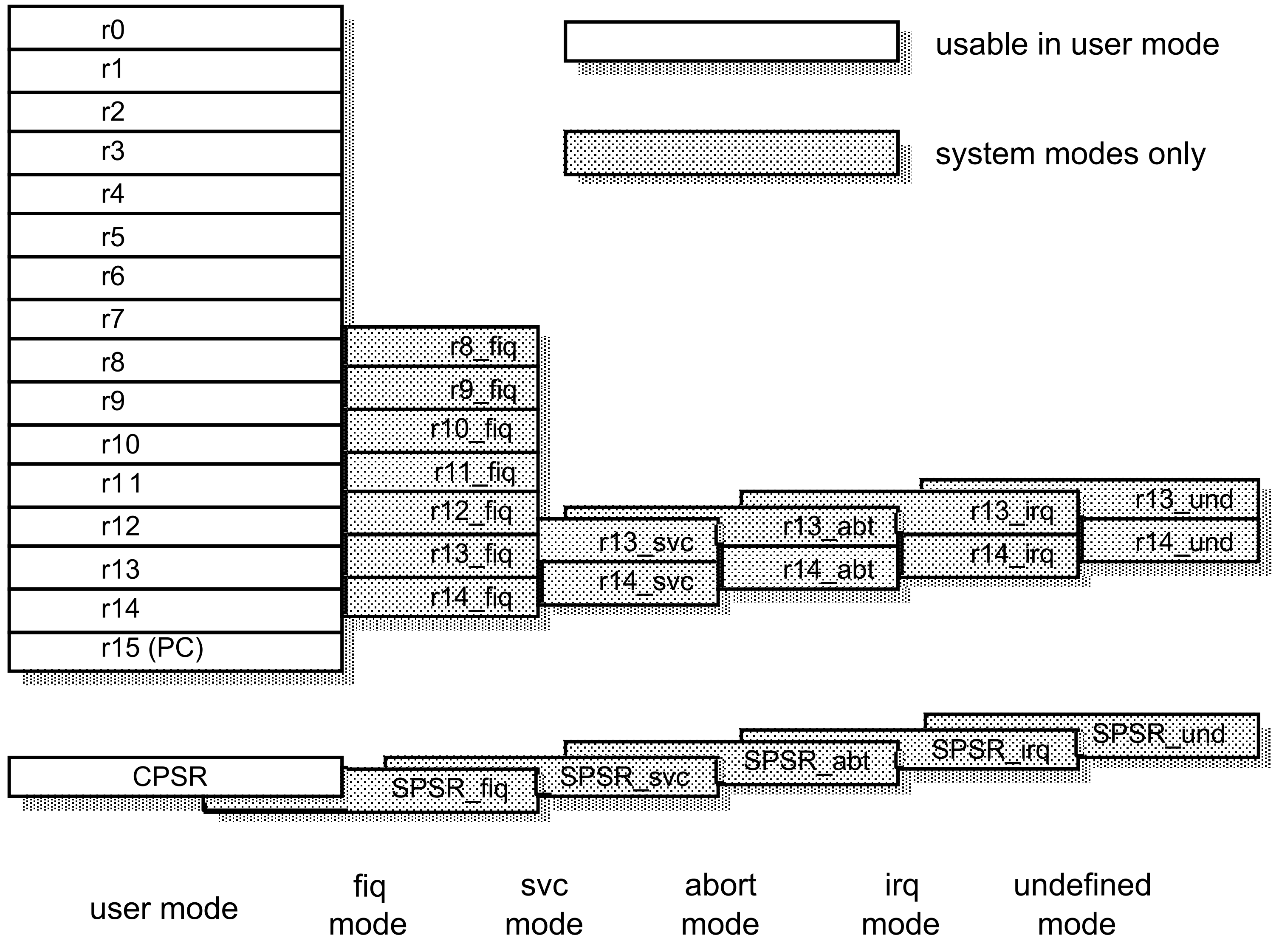
- **Multi-ICE connection**



Basic Debug Requirements

- Control of program execution
  - set watchpoints on interesting data accesses
  - set breakpoints on interesting instructions
  - single step through code
- Examine and change processor state
  - view and set register values
- Examine and change system state
  - access system memory
    - download initial code
- Interleaving source code
  - show C/C++ code and disassembly code together

Register Banking



Outline

- Debugging skills
- *Software Quality Measurement*

Software Quality Measurement

- Memory requirement of the program
  - Memory type: volatile (RAM), non-volatile (ROM)
  - Memory performance: access speed, data width, size and range
- Profiling
  - build up a picture of the percentage of time spent in each procedure.
- Benchmarking performance
  - Evaluate software performance prior to implement on hardware
- Writing efficient C for ARM cores
  - ARM/Thumb interworking
  - Coding styles

Application Code and Data Size

armlink offers two options to provide the relevant information:

-info sizes (sizes of all objects)

-info totals (summary only)

```
==============================================================
Image component sizes
   Code    RO Data    RW Data    ZI Data      Debug
  25840       3444          0          0     104344   Object Totals
  22680        762          0        300       9104    Library Totals
===============================================================
   Code    RO Data    RW Data    ZI Data      Debug
  48520       4206          0        300     113448   Grand Totals
===============================================================
   Total RO  Size(Code + RO Data)                          52726 (  51.49kB)
   Total RW  Size(RW Data + ZI Data)                       300 (   0.29kB)
   Total ROM Size(Code + RO Data + RW Data)                52726 (  51.49kB)
===============================================================
```

- The size of code/data in
  - an ELF image can be viewed using fromelf –z
  - a library can be viewed using armar –sizes

ARM and Thumb Code Size

Simple C routine

```
if (x>=0)
    return x;
else
    return -x;
```

The equivalent ARM assembly

```
labs    CMP     r0,#0       ;Compare r0 to zero
        RSBLT   r0,r0,#0    ;If r0<0 (less than=LT) then do r0= 0-r0
        MOV     pc,lr       ;Move Link Register to PC (Return)
```

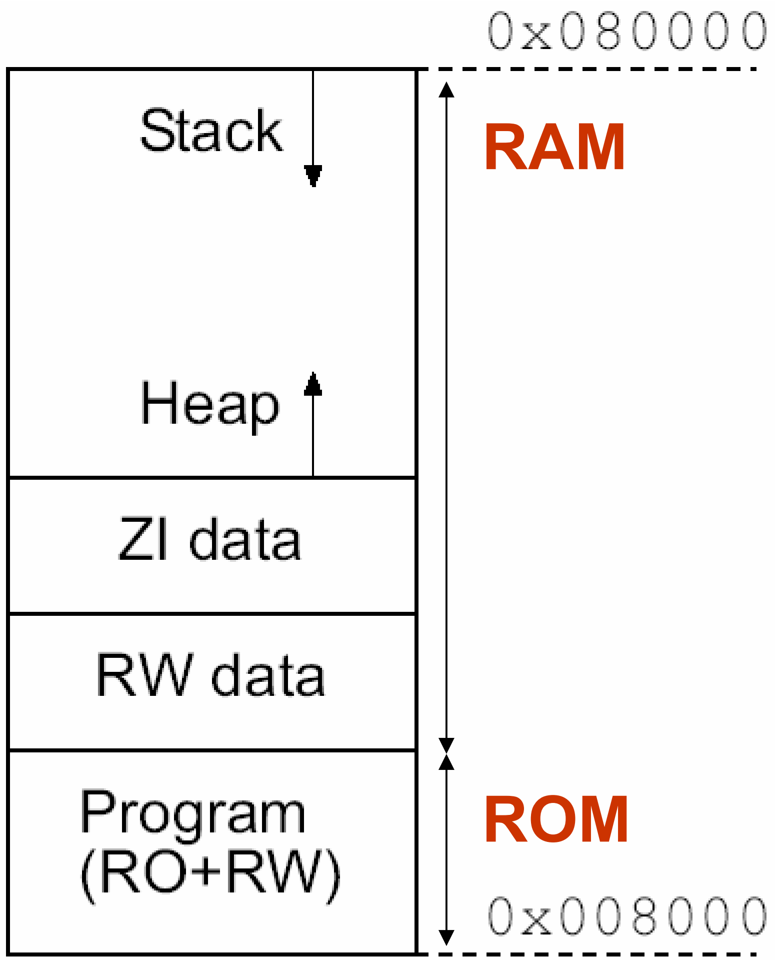
The equivalent Thumb assembly

```
        CODE16 ;Directive specifying 16-bit (Thumb) instructions
labs    CMP     r0,#0       ;Compare r0 to zero
        BGE     return      ;Jump to Return if greater or
                            ;equal to zero
        NEG     r0,r0       ;If not, negate r0
return  MOV     pc,lr       ;Move Link register to PC (Return)
```

Code	Instructions	Size (Bytes)	Normalised
ARM	3	12	1.0
Thumb	4	8	0.67

Memory Map and Size Considerations

- The linker calculates the ROM and RAM requirements for code and data as follows:
  - **ROM**: Code size + RO data + RW data
  - **RAM**: RW Data + ZI data.
- You may wish to copy **code** from **ROM** into faster RAM, which will also increase the **RAM** requirements
- 32-bit memory on-chip will significantly improve over 8 or 16- bit off-chip memory

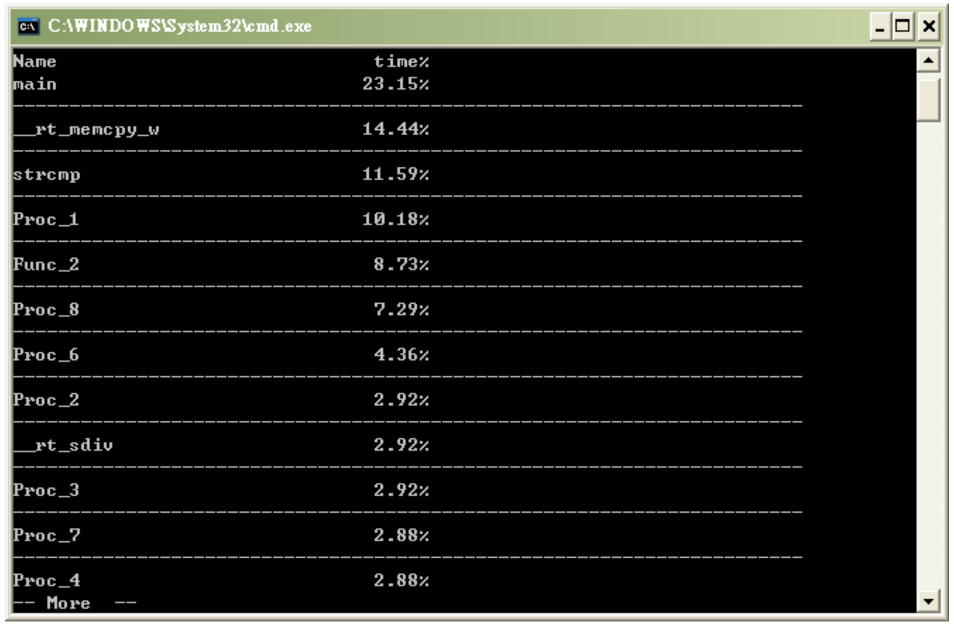


Default memory map

Profiling (1/3)

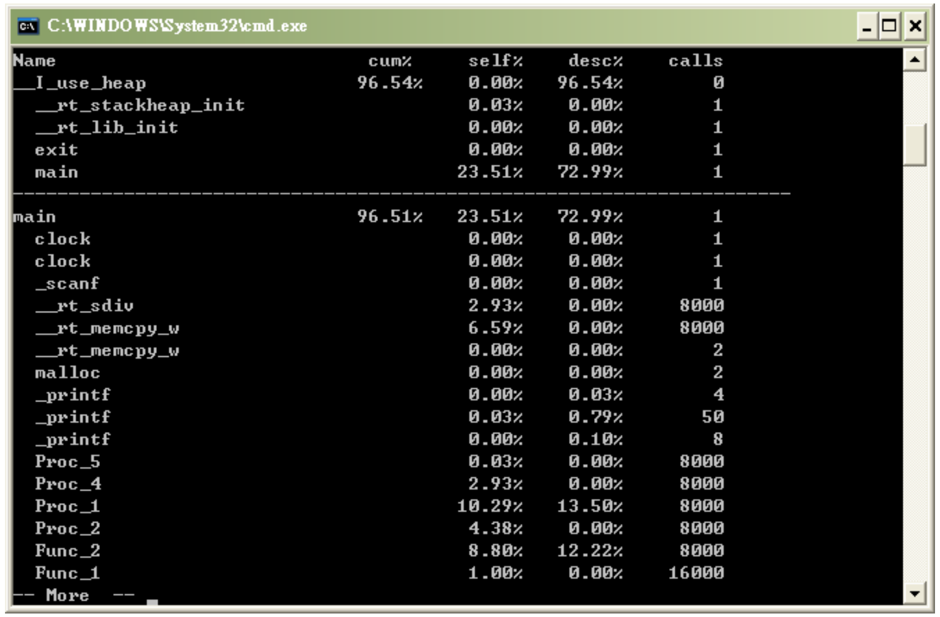
- About Profiling:
  - Profiler samples the **program counter** and computes the percentage time of each function spent.
  - **Flat Profiling**:
    - If only pc-sampling info. is present. It can only display the time percentage spent in each function excluding the time in its children.
    - Flat profiling accumulates limited information without altering the image
  - **Call graph Profiling**:
    - If function call count info. is present. It can show the approximations of the time spent in each function including the time in its children.
    - Extra code is added to the image

Profiling (2/3)



```
C:\WINDOWS\System32\cmd.exe
Name                                    time%
main                                   23.15%
---------------------------------------------------------------------------
__rt_memcpy_w                          14.44%
---------------------------------------------------------------------------
strcmp                                 11.59%
---------------------------------------------------------------------------
Proc_1                                 10.18%
---------------------------------------------------------------------------
Func_2                                  8.73%
---------------------------------------------------------------------------
Proc_8                                  7.29%
---------------------------------------------------------------------------
Proc_6                                  4.36%
---------------------------------------------------------------------------
Proc_2                                  2.92%
---------------------------------------------------------------------------
__rt_sdiv                               2.92%
---------------------------------------------------------------------------
Proc_3                                  2.92%
---------------------------------------------------------------------------
Proc_7                                  2.88%
---------------------------------------------------------------------------
Proc_4                                  2.88%
-- More  --
```

Flat Profiling



```
C:\WINDOWS\System32\cmd.exe
Name                             cum%     self%    desc%     calls
__I_use_heap                    96.54%    0.00%   96.54%         0
  __rt_stackheap_init                     0.03%    0.00%         1
  __rt_lib_init                           0.00%    0.00%         1
  exit                                    0.00%    0.00%         1
  main                                   23.51%   72.99%         1
----------------------------------------------------------------------
main                            96.51%   23.51%   72.99%         1
  clock                                   0.00%    0.00%         1
  clock                                   0.00%    0.00%         1
  _scanf                                  0.00%    0.00%         1
  __rt_sdiv                               2.93%    0.00%      8000
  __rt_memcpy_w                           6.59%    0.00%      8000
  __rt_memcpy_w                           0.00%    0.00%         2
  malloc                                  0.00%    0.00%         2
  _printf                                 0.00%    0.03%         4
  _printf                                 0.03%    0.79%        50
  _printf                                 0.00%    0.10%         8
  Proc_5                                  0.03%    0.00%      8000
  Proc_4                                  2.93%    0.00%      8000
  Proc_1                                 10.29%   13.50%      8000
  Proc_2                                  4.38%    0.00%      8000
  Func_2                                  8.80%   12.22%      8000
  Func_1                                  1.00%    0.00%     16000
-- More  --
```

Call graph Profiling

- Profiling Limitations:
  - Profiling is NOT available for code in ROM, or for scatter loaded images.
  - No data is gathered for programs that are too small.

Profiling (3/3)

- The Profiler command syntax is as follows:

armprof [-parent|-noparent] [-child|-nochild] [-sort options] prf_file_name

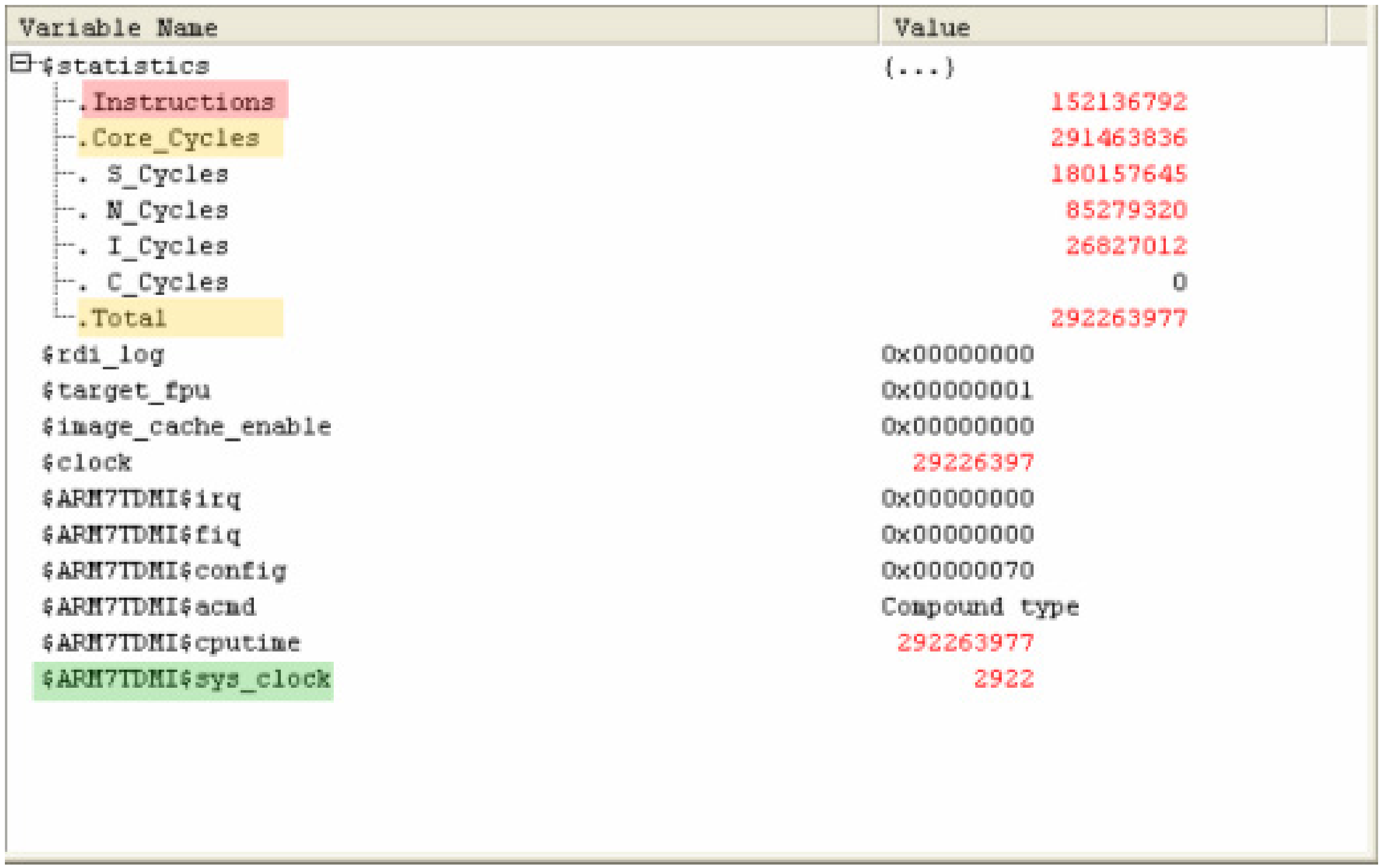
cumulative
self
descendants
calls

- Call graph Profiling Sample Output

```
Name                 cum%   self%  desc%  calls
----------------------------------------------------------------
 main                       17.69% 60.06% 1
insert_sort          77.76% 17.69% 60.06% 1
 strcmp                     60.06% 0.00%  243432
----------------------------------------------------------------
 qs_string_compare          3.21%  0.00%  13021
 shell_sort                 3.46%  0.00%  14059
 insert_sort                60.06% 0.00%  243432
strcmp               66.75% 66.75% 0.00%  270512
----------------------------------------------------------------
```

Performance benchmarking (1/5)

- **Execution time (real-time vs. emulated)**
  - $sys_clock
  - Execution time = Total Cycle count * Cycle time



```
Variable Name                    Value
⊟$statistics                     {...}
  .Instructions                           152136792
  .Core_Cycles                            291463836
  . S_Cycles                              180157645
  . N_Cycles                               85279320
  . I_Cycles                               26827012
  . C_Cycles                                      0
  .Total                                  292263977
 $rdi_log                        0x00000000
 $target_fpu                     0x00000001
 $image_cache_enable             0x00000000
 $clock                            29226397
 $ARM7TDMI$irq                   0x00000000
 $ARM7TDMI$fiq                   0x00000000
 $ARM7TDMI$config                0x00000070
 $ARM7TDMI$acmd                  Compound type
 $ARM7TDMI$cputime                292263977
 $ARM7TDMI$sys_clock                   2922
```

Performance benchmarking (2/5)

- When ARM processor executes program, it will change these clock types according to the demand of operating.
- Cycle Types (Von Neuman Cores) for ARM7TDMI
  - N-cycles (Non-sequential cycle)
    - The ARM core requests a transfer to or from an address which is unrelated to the address used in the preceding cycle.
  - S-cycles (Sequential cycle)
    - The ARM core requests a transfer to or from an address which is either the same, or one word or one-half-word greater than the preceding address.
  - I-cycles (Internal cycle)
    - The ARM core does not require a transfer, as it is performing an internal function.
  - C-cycles (Coprocessor register transfer cycle)
    - The ARM core wishes to use the data bus to communicate with a coprocessor, but does not require any action by the memory system
  - Total clock cycle = (N + S + I + C + Waits)-cycles

Performance benchmarking (3/5)

- Cycle Types (Harvard Cores) for ARM9TDMI

Cycle types	Instruction bus	Data bus	Meaning
Core-cycles	-	-	The total number of core clock. (including pipeline stalls due to interlocks and multiple cycle instructions)
ID-cycles	Active	Active	-
I-cycles	Active	Idle	-
D-cycles	Idle	Active	-
Idle-cycles	Idle	Idle	-

– Total clock cycle = (Core + ID + I + D + idle + Waits)-cycles

Performance benchmarking (4/5)

Estimation using different Memory model

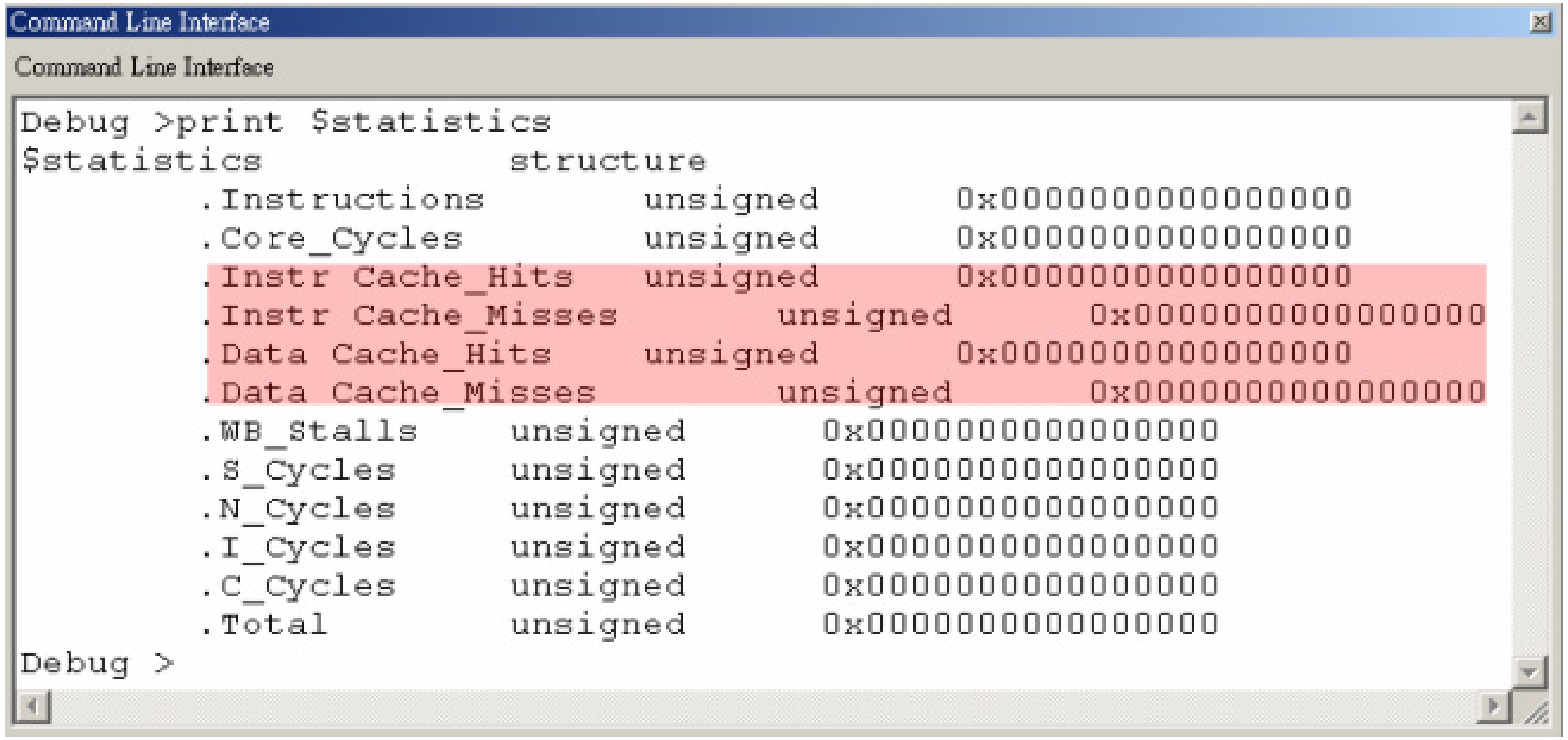
- If no map file is specified:
  - ARMulator will use a 4GB bank of 'ideal' memory, i.e., no wait states.
- The map file defines regions of memory, and, for each region:
  - The address range
  - The data bus width (in bytes).
  - The access times (in ns)
- armsd.map typically contains memory map information:

					read time	write time
start address	length	label	width	access	non-s/seq	non-s/seq
00000000	00020000	ROM	2	R	150/100	150/100
10000000	00008000	RAM	4	RW	100/65	100/65

Performance benchmarking (5/5)

Benchmarking cached cores

- Cache efficiency
  - Avg. memory access time = hit time +Miss rate * Miss Penalty
  - Cache Efficiency = Core-Cycles / Total Bus Cycles



```
Debug >print $statistics
$statistics                structure
        .Instructions         unsigned      0x0000000000000000
        .Core_Cycles          unsigned      0x0000000000000000
        .Instr Cache_Hits     unsigned      0x0000000000000000
        .Instr Cache_Misses         unsigned      0x0000000000000000
        .Data Cache_Hits      unsigned      0x0000000000000000
        .Data Cache_Misses          unsigned      0x0000000000000000
        .WB_Stalls      unsigned      0x0000000000000000
        .S_Cycles       unsigned      0x0000000000000000
        .N_Cycles       unsigned      0x0000000000000000
        .I_Cycles       unsigned      0x0000000000000000
        .C_Cycles       unsigned      0x0000000000000000
        .Total          unsigned      0x0000000000000000
Debug >
```

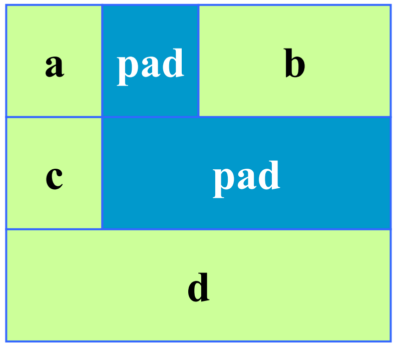
Efficient Code Development

- **ARM/Thumb interworking**
  - ARM : bottleneck, interrupt handle
  - Thumb: code size
- **Compiler optimization:**
  - Space or speed (e.g, **-Ospace(default)** or **-Otime**)
  - Effort (e.g., **-O0** ,**-O1** or **-O2**)
  - Instruction scheduling
- **Coding style**
  - Variable type and size
  - Parameter passing
  - Loop termination
  - Division operation and modulo arithmetic

Data Layout

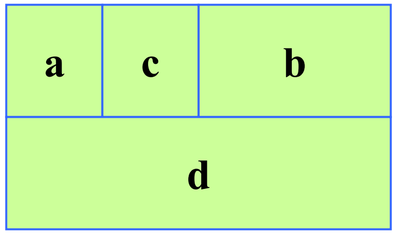
Default

```
char a;
short b;
char c;
int d;
```



Optimized

```
char a;
char c;
short b;
int d;
```



occupies 8 bytes, without any padding

occupies 12 bytes, with 4 bytes of padding

- **Group variables of the same type together**
  - **The best way to ensure that as little padding data as possible is added by the compiler.**

Variable Types – Size Examples

```
int intinc
(int a)
{ return a + 1;
}
```

```
intinc
ADD a1,a1,#1
MOV pc,lr
```

```
short shortinc
(short a)
{ return a + 1;
}
```

```
shortinc
ADD a1,a1,#1
MOV a1,a1,LSL #16
MOV a1,a1,ASR #16
MOV pc,lr
```

```
char charinc (char
a)
{ return a + 1;
}
```

```
charinc
ADD a1,a1,#1
AND a1,a1,#&ff
MOV pc,lr
```

Stack Usage

- Stack usage:
  - Return addresses for subroutines
  - Local arrays & structures
- To minimize stack usage:
  - Keep functions small (few variables, less spills)minimize the number of 'live' variables (i.e., those which contain useful data at each point in the function)
  - Avoid using large local structures or arrays (use dynamic allocation malloc/free instead)
  - Avoid recursion

Global Data Issues

- How much **space the variables occupy at run time.**
  - This determines the **size of RAM** required for a program to run. The ARM compilers may insert padding bytes between variables, to ensure that they are properly aligned.
- How much **space the variables occupy in the image**.
  - This is one of the factors determining the **size of ROM** needed to hold a program. Some global variables which are not explicitly initialized in your program may nevertheless have their initial value (of zero, as defined by the C standard) stored in the image.
- How many **codes need to access the variables**.
  - Some data organizations require more code to access the data. As an extreme example, the smallest data size would be achieved if all variables were stored in suitably sized bitfields, but the code required to access them would be much larger (trade-off between size and performance)

Loop termination

```
…
int acc(int n) {
  int i;           //loop index
  int sum=0;

  for (i=1; i<=n ;i++)
    sum+=i;
  return sum;

}
…
```

loop.c

```
…
int acc(int n) {
  int i;           //loop index
  int sum=0;

  for (i=n; i!=0 ;i--)
    sum+=i;
  return sum;

}
…
```

loop_opt.c

Division operation and modulo arithmetic

- **The remainder operator '%' is commonly used in modulo arithmetic.**
  - This will be expensive if the modulo value is **not a power of two**
  - This can be avoid by rewriting C code to use **if ()** statement heck

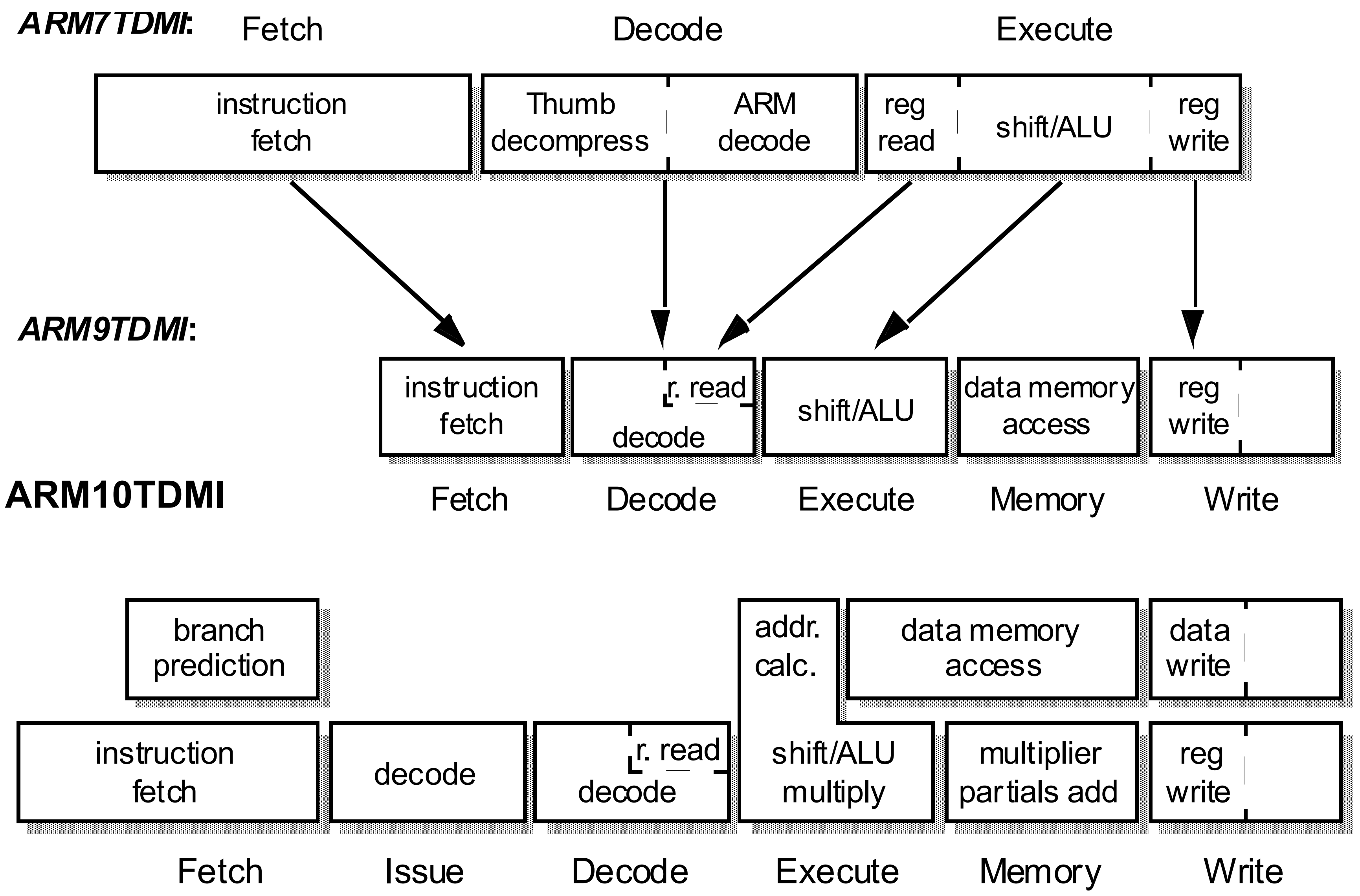
```
unsigned counter1 (unsigned
counter)
{ return (++counter % 60);
}
==============================
counter1
    STMFB    sp!, {lr}
    ADD      r1, r0, #1
    MOV      r0, #0x3C
    BL       __rt_udiv
    MOV      r0, r1
    LDMIA    sp!, {pc}
```

modulo.c

```
unsigned counter2 (unsigned
counter)
{ if (++counter >= 60)
      counter=0;
   return counter
}
==============================
counter2
    ADD      r0, r0, #1
    CMP      r0, #0x3C
    MOVCS    r0, #0
    MOV      pc, lr
```

modulo_opt.c

ARM Processor Pipeline Operations



Dhrystone Result Example

Target: ARM940T, 4kB I-cache, 4kB D-cache, 10.00MHz core clock, (Physical memory, 3.3MHz)

	Instructions	Core Cycles	S-cycles	N-cycles	I-cycles	C-cycles	Total
Iteration 1	306	446	377	0	345	0	722
Iteration n	306	446	7	0	142	0	149

Iteration 1: 673 x 1 / 3,333,333 = 216.6us

Iteration n: 149 x 1 / 3,333,333 = 44.7us

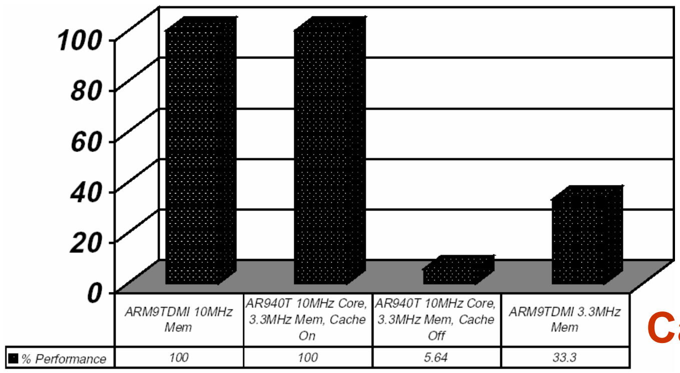
446/149 = 2.993

Iteration 1~n: Total Core Cycles: 27074407

Total Bus Cycles : 9034428

Cache Efficiency : 2.9979 (MCCFG=3)

Cache Efficiency % : 100 x (Cache Efficiency x MCCFG) = 99.93%

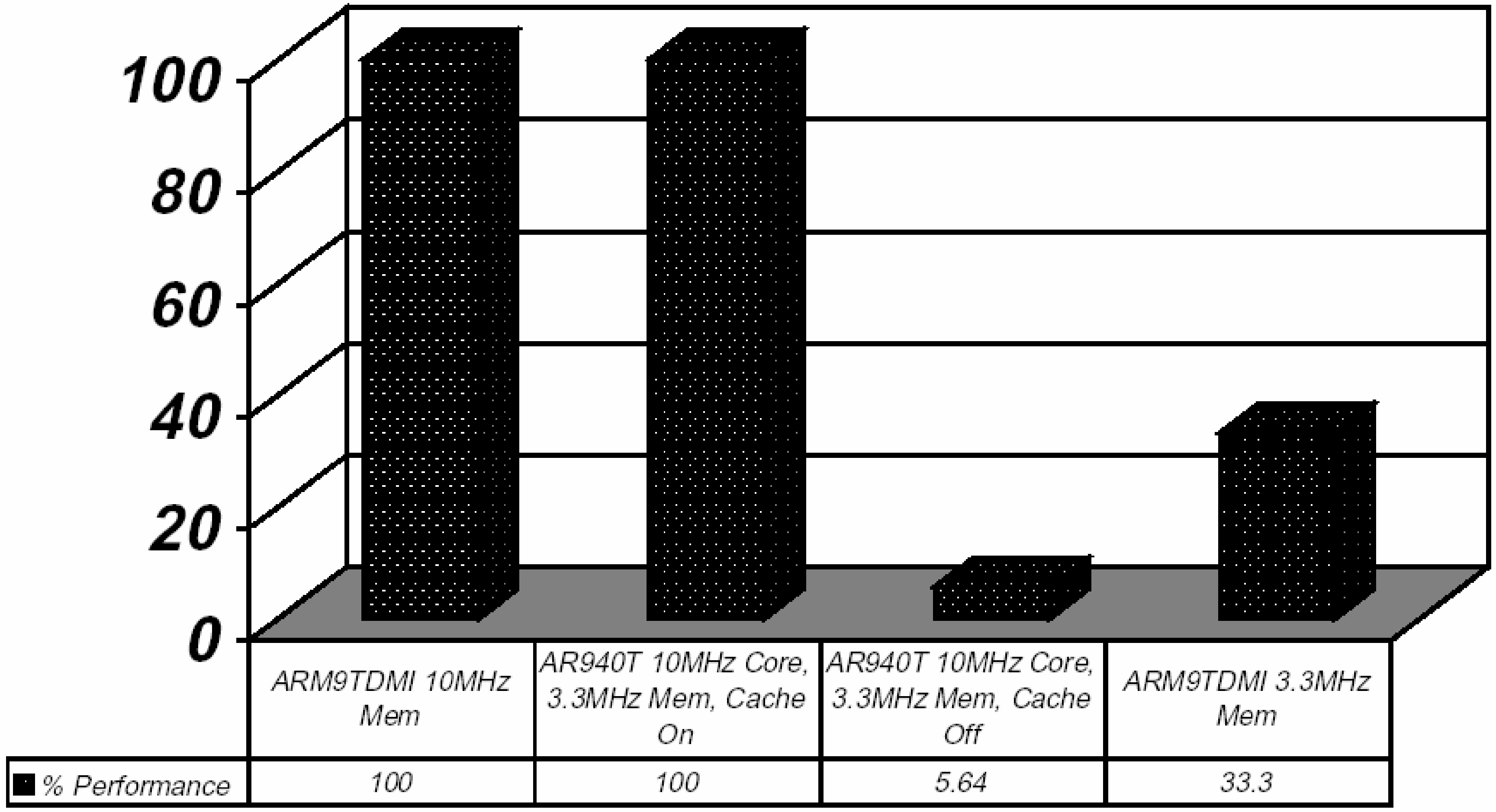


	ARM9TDMI 10MHz Mem	AR940T 10MHz Core, 3.3MHz Mem, Cache On	AR940T 10MHz Core, 3.3MHz Mem, Cache Off	ARM9TDMI 3.3MHz Mem
■ % Performance	100	100	5.64	33.3

Cached with different clock domains

Dhrystone Analysis

Cached with different clock domains



	ARM9TDMI 10MHz Mem	AR940T 10MHz Core, 3.3MHz Mem, Cache On	AR940T 10MHz Core, 3.3MHz Mem, Cache Off	ARM9TDMI 3.3MHz Mem
■ % Performance	100	100	5.64	33.3

Lab 2: Debugging and Evaluation

- Goal
  - How to perform a variety of debugging tasks and software quality evaluation
  - How to profile and evaluation the software performance of the application
- Guidance
  - Instruction of this lab
  - Preconfigured project stationery files
- Steps
  - Debugging skills
  - Software Quality Measurement
- Requirements and Exercises
  - See note file
- Discussion
  - The approaches to minimize the code size and enhance the performance of the program
  - The advantages of ARM/Thumb instruction sets interworking