ARM Assembly

Computer Architecture

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CS3501 – 2024 II
PROF.: JGONZALEZ@UTEC.EDU.PE
CWILLIAMS@UTEC.EDU.PE
SRC: HARRIS, HARRIS - DIGITAL DESIGN AND COMPUTER ARCHITECTURE

Executive Summary

- Motivation: Programs at high-level languages can be executed in modern processor systems.
- Problem: We need to understand the machine code and define its relationship with programming languages constructs.
- Overview:
  - Overview of conditional statements, loops, arrays and function calls.
  - ARM Assembly programming for high-level constructs.
  - Code and execute with an ARM emulator.
- Conclusion: We can create complex programs using assembly language by defining correct machine code instructions.

Resources

ARM Emulators

- **CPUlator (for course lab):** **https://cpulator.01xz.net/?sys=arm**
  Web-based, online simulator.
- **ARM Visual (optional):** https://salmanarif.bitbucket.io/visual/
  Windows, MacOS, Ubuntu

ARM Quick Reference Guide: **https://developer.arm.com/documentation/qrc0001/m**

ARM Cheat Sheet by Uwe Zimmer:
https://cs.anu.edu.au/courses/comp2300/v_media/manuals/ARMv7-cheat-sheet.pdf

Outline

Conditional Statements

Loops

Arrays

Conclusions

if Statement

C Code

```
if (i == j)
  f = g + h;



f = f - i;
```

ARM Assembly Code

if Statement

C Code

```
if (i == j)
  f = g + h;



f = f - i;
```

ARM Assembly Code

```
;R0=f, R1=g, R2=h, R3=i, R4=j

 CMP R3, R4        ; set flags with R3-R4
 BNE L1            ; if i!=j, skip if block
 ADD R0, R1, R2    ; f = g + h

L1
 SUB R0, R0, R2    ; f = f - i
```

Assembly tests opposite case (`i != j`) of high-level code (`i == j`)

if Statement: Alternate Code

C Code

```
if (i == j)
  f = g + h;
f = f - i;
```

ARM Assembly Code

```
;R0=f, R1=g, R2=h, R3=i, R4=j

 CMP   R3, R4      ; set flags with R3-R4
 ADDEQ R0, R1, R2  ; if (i==j) f = g + h
 SUB   R0, R0, R2  ; f = f - i
```

if Statement: Alternate Code

Original

```
 CMP R3, R4
 BNE L1
 ADD R0, R1, R2
L1
 SUB R0, R0, R2
```

Alternate Assembly Code

```
;R0=f, R1=g, R2=h, R3=i, R4=j

 CMP   R3, R4    ; set flags with R3-R4
 ADDEQ R0, R1, R2  ; if (i==j) f = g + h
 SUB   R0, R0, R2  ; f = f - i
```

Useful for **short** conditional blocks of code

if/else Statement

C Code

```
if (i == j)
  f = g + h;


else
  f = f - i;
```

ARM Assembly Code

```
;R0=f, R1=g, R2=h, R3=i, R4=j
```

if/else Statement

C Code

```
if (i == j)
  f = g + h;


else
  f = f - i;
```

ARM Assembly Code

```
;R0=f, R1=g, R2=h, R3=i, R4=j

 CMP R3, R4      ; set flags with R3-R4
 BNE L1          ; if i!=j, skip if block
 ADD R0, R1, R2  ; f = g + h
 B   L2          ; branch past else block
L1
 SUB R0, R0, R2  ; f = f - i
L2
```

if/else Statement: Alternate Code

C Code

```
if (i == j)
  f = g + h;
else
   f = f - i;
```

ARM Assembly Code

```
;R0=f, R1=g, R2=h, R3=i, R4=j

 CMP   R3, R4      ; set flags with R3-R4
 ADDEQ R0, R1, R2   ; if (i==j) f = g + h

SUBNE R0, R0, R2   ; else f = f - i
```

if/else Statement: Alternate Code

Original

```
 CMP R3, R4
 BNE L1
 ADD R0, R1, R2
 B   L2
L1
 SUB R0, R0, R2
L2
```

Alternate Assembly Code

```
;R0=f, R1=g, R2=h, R3=i, R4=j

 CMP   R3, R4     ; set flags with R3-R4
 ADDEQ R0, R1, R2  ; if (i==j) f = g + h

SUBNE R0, R0, R2  ; else f = f - i
```

Outline

Conditional Statements

Loops

Arrays

Conclusions

while Loops

C Code

```
// determines the power
// of x such that 2^x = 128
int pow = 1;
int x   = 0;


while (pow != 128) {

  pow = pow * 2;
  x = x + 1;
}
```

ARM Assembly Code

```
; R0 = pow, R1 = x
```

while Loops

C Code

```
// determines the power
// of x such that 2^x = 128
int pow = 1;
int x   = 0;


while (pow != 128) {

  pow = pow * 2;
  x = x + 1;
}
```

ARM Assembly Code

```
; R0 = pow, R1 = x
  MOV      R0, #1                ; pow = 1
  MOV      R1, #0                ; x = 0

WHILE
  CMP R0, #128                   ; R0-128
  BEQ DONE                       ; if (pow==128)
                                 ; exit loop
  LSL R0, R0, #1     ; pow=pow*2
  ADD R1, R1, #1     ; x=x+1
  B   WHILE                      ; repeat loop

DONE
```

Assembly tests for the opposite case (`pow == 128`) of the C code (`pow != 128`).

for Loops

```
for (initialization; condition; loop operation)
   statement
```

- **initialization:** executes before the loop begins
- **condition:** is tested at the beginning of each iteration
- **loop operation:** executes at the end of each iteration
- **statement:** executes each time the condition is met

for Loops

C Code

```
// adds numbers from 1-9
int sum = 0


for (i=1; i!=10; i=i+1)
   sum = sum + i;
```

ARM Assembly Code

```
; R0 = i, R1 = sum
```

for Loops

C Code

```
// adds numbers from 1-9
int sum = 0


for (i=1; i!=10; i=i+1)
   sum = sum + i;
```

ARM Assembly Code

```
; R0 = i, R1 = sum
  MOV     R0, #1                ; i = 1
  MOV     R1, #0                ; sum = 0

FOR
  CMP R0, #10                   ; R0-10
  BEQ DONE                      ; if (i==10)
                                ; exit loop
  ADD R1, R1, R0     ; sum=sum + i
  ADD R0, R0, #1     ; i = i + 1
  B   FOR            ; repeat loop

DONE
```

for Loops: Decremented Loops

In ARM, decremented loop variables are more efficient

C Code

```
// adds numbers from 1-9
int sum = 0


for (i=9; i!=0; i=i-1)
   sum = sum + i;
```

ARM Assembly Code

```
; R0 = i, R1 = sum
  MOV        R0, #9       ; i = 9
  MOV        R1, #0       ; sum = 0

FOR
  ADD  R1, R1, R0         ; sum=sum + i
  SUBS R0, R0, #1         ; i = i − 1
                          ; and set flags
  BNE  FOR                ; if (i!=0)
                          ; repeat loop
```

Saves 2 instructions per iteration:

- Decrement loop variable & compare: `SUBS R0, R0, #1`
- Only 1 branch, instead of 2

Outline

Conditional Statements

Loops

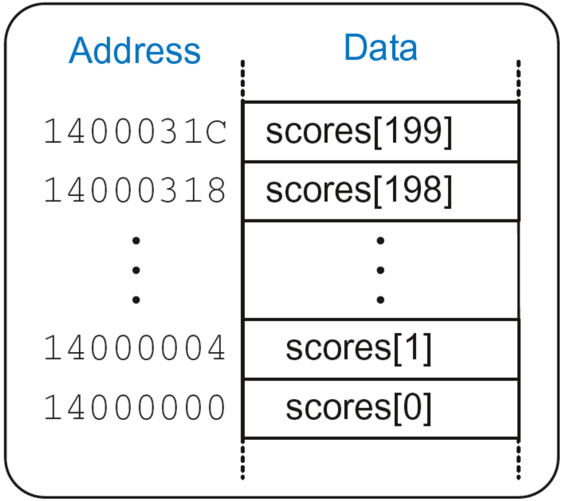
Arrays

Conclusions

Arrays

- Access large amounts of similar data
  - **Index:** access to each element
  - **Size:** number of elements

- **Example:** 5-element array
  - **Base address** = 0x14000000 (address of first element, scores[0])
  - Array elements accessed relative to base address



Address	Data
1400031C	scores[199]
14000318	scores[198]
⋮	⋮
14000004	scores[1]
14000000	scores[0]

Main memory

Accessing Arrays

C Code

```
  int array[5];
  array[0] = array[0] * 8;
  array[1] = array[1] * 8;
```

ARM Assembly Code

```
; R0 = array base address
  MOV R0, #0x60000000          ; R0 = 0x60000000

  LDR R1, [R0]                 ; R1 = array[0]
  LSL R1, R1, 3                ; R1 = R1 << 3 = R1*8
  STR R1, [R0]                 ; array[0] = R1

  LDR R1, [R0, #4]             ; R1 = array[1]
  LSL R1, R1, 3                ; R1 = R1 << 3 = R1*8
  STR R1, [R0, #4]             ; array[1] = R1
```

Arrays using for Loops

C Code

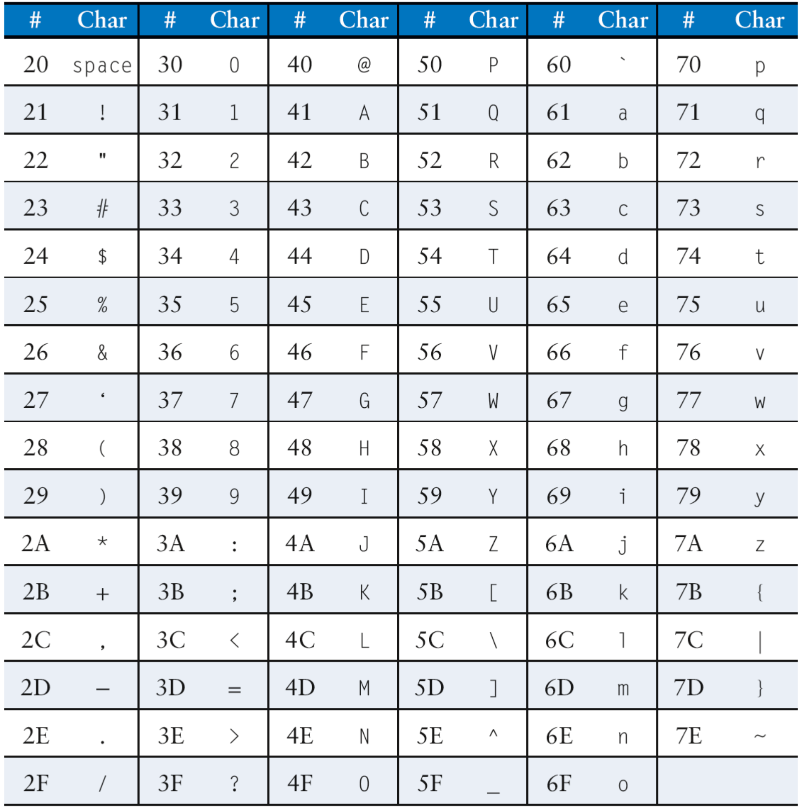
```
int array[200];
int i;

for (i=199; i >= 0; i = i - 1)
        array[i] = array[i] * 8;
```

ARM Assembly Code

```
; R0 = array base address, R1 = i
  MOV R0, #0x60000000
  MOV R1, #199
FOR
  LDR  R2, [R0, R1, LSL #2]; R2 = array(i)
  LSL  R2, R2, #3          ; R2 = R2<<3 = R3*8
  STR  R2, [R0, R1, LSL #2]; array(i) = R2
  SUBS R0, R0, #1          ; i = i - 1
                          ; and set flags
  BPL  FOR                ; if (i>=0) repeat loop
```

ASCII Code and Cast of Characters

- American Standard Code for Information Interchange (ASCII)
- Each text character has unique byte value
  - For example, S = 0x53, a = 0x61, A = 0x41
  - Lower-case and upper-case differ by 0x20 (32)



#	Char	#	Char	#	Char	#	Char	#	Char	#	Char
20	space	30	0	40	@	50	P	60	`	70	p
21	!	31	1	41	A	51	Q	61	a	71	q
22	"	32	2	42	B	52	R	62	b	72	r
23	#	33	3	43	C	53	S	63	c	73	s
24	$	34	4	44	D	54	T	64	d	74	t
25	%	35	5	45	E	55	U	65	e	75	u
26	&	36	6	46	F	56	V	66	f	76	v
27	‘	37	7	47	G	57	W	67	g	77	w
28	(	38	8	48	H	58	X	68	h	78	x
29	)	39	9	49	I	59	Y	69	i	79	y
2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
2D	–	3D	=	4D	M	5D	]	6D	m	7D	}
2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
2F	/	3F	?	4F	O	5F	_	6F	o		

Outline

Conditional Statements

Loops

Arrays

Conclusions

Conclusions

- **We reviewed fundamentals concepts in programming languages constructs.**
- **We reviewed** assembly implementation for conditional statements, loops, arrays.
- **We coded and executed high-level constructs using ARM syntax and instructions.**
- We conclude that **complex programs have a direct implementation in machine code that allows to execute them in the processor.**

ARM Assembly

Computer Architecture

UTEC
UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA

CS3501 – 2024II
PROF.: JGONZALEZ@UTEC.EDU.PE
CWILLIAMS@UTEC.EDU.PE