Rekenaarstelsels 245 - Prakties 2
Computer Systems 245 - Practical 2

2015-08-07

1 Sommeerder in ASM[1]

Werk in groepe van 2 saam. Elkeen moet die prakties inhandig, maar al die groeplede se name moet bo aan dit verskyn. Die name kan as 'n kommentaarstelling bo aand die kode ingetik word.

Gebruik die kode wat aan jou verskaf is op `learn.sun.ac.za`. Net soos in vorige week se prakties is daar 'n RSPrak2.c-lêer wat funksies roep wat in 'n .asm-lêer gedefiniëer is. Hierdie week is daar twee .asm-lêers (adder.asm en changecase.asm). Skep 'n nuwe projek en voeg die bron lêers by. Vir hierdie vraag voltooi die funksie `_adder` in die lêer `adder.asm`. Dit ontvang 4 integers vanuit C, moet dit by mekaar tel en weer terugstuur na C.

Om hierdie vraag te kan voltooi sal jy moet verwys na **RL78-ABI-Specification** en **RL78 software manual (asm)** wat beide lê onder "Dokumentasie" op `learn.sun.ac.za`. Dit sal 'n goeie idee wees om die bladsye wat jy nodig het in die sagtewarehandleiding uit te druk.

Gebruik e2Studio se register-, veranderlike- en geheue-inspekteurs om te sien wat gebeur en wat jou antwoord is. Stel breekpunte op gerieflike plekke in jou kode om die stand van jou registers en veranderlikes op daardie punt te sien.

Gee jou voltooide **adder.asm**-lêer in op `learn.sun.ac.za` voor **vanmiddag 17:00**. Dit sal wys wees om dit te voltooi en in te gee voor jy aangaan met die volgende vraag. As jy hierdie vraag nie betyds klaarkry nie, gee dit in so ver soos jy gekom het. *Verwyder jou kode vanaf die labrekenaars aan die einde van die prakties sodat ander mense dit nie kan plagieer nie.*

Verwys na afdeling 3 vir 'n paar wenke.

'n Algemene reël is: Hoe meer ASM instruksies 'n mens se program gebruik, hoe langer sal dit neem om uit te voer. Probeer daarom enige ASM kode wat jy skryf so kort as moontlik te maak.

1 Adder in ASM

Work in groups of 2. Everyone needs to hand in the practical, but with all the group members' names on it. The names can by typed as a comment inside the code.

Use the code that is provided to you on `learn.sun.ac.za`. Just like last week, there is a RSPrak.c file that calls functions from an .asm file. This week there are two .asm files (adder.asm and changecase.asm). `adder.asm` contains the function called `_adder` you need to complete for this question. It receives 4 integers as parameters from C, need to add them together and return the result back to C.

To complete this question, you will need to refer to **RL78-ABI-Specification** and **RL78 software manual (asm)** which are both available under "Documentation" on `learn.sun.ac.za`. It will be a good idea to print out the pages yoou need from the Software manual.

Use e2Studio's built in register, variable and memory inspectors to see what is going on in your code and to see the answer. Set break points in your code to inspect the content of registers and variables at that point.

Hand in your completed **adder.asm** file on `learn.sun.ac.za` before **today at 17:00**. It would be wise to complete this question and hand it in before you continue with the following question. If you can't complete it in time, hand in what you have. *Remove all your code from the lab computers after the practical so that other people can't copy it.*

See section 3 for a few hints.

A general rule for ASM is: The more instructions you use, the longer the program will take to execute. Therefore always try to make your ASM program as short as possible.

[1] Dit is makliker om die afkorting **ASM** te gebruik as om elke keer **saamsteltaal** uit te skryf.

2 Teksbewerking in ASM

Voltooi die funksie `_verander_kas` in `changecase.asm`. Die funksie moet kleinletter alfabetiese karakters omskakel na hoofletters, en hoofletters omskakel na kleinletters. Alle ander karakters moet onveranderd gelos word.

Wenk: Skryf die ASCII waardes van a, z, A en Z uit in binêr. Met hoeveel verskil 'n hoof- en kleinletter? Teken dan 'n vloeidiagram van hoe jou program sal vloei, waardes sal toets en spring om deur die sin te stap.

- C stuur slegs die geheue-adres van die eerste karakter oor as 'n parameter. Dit is 'n 16 bis getal.
- C plaas 'n NUL-waarde karakter aan die einde van 'n sin (ASCIIZ standaard).

Handig jou `changecase.asm`-lêer in op `learn.sun.ac.za` voor **23:55 op Donderdag 13 Augustus**.

2 Text manipulation in ASM

Complete the function `_verander_kas` in `changecase.asm`. The function should convert lower case characters to upper case, and all upper case characters to lower case characters. All other characters must remain unchanged.

Hint: Write the ASCII values of a,z, A and Z out in binary. Draw a flow diagram showing the program flow, conditions and jumps the program will use to step through the sentence.

- C will only send the address of the string and the address is a 16 bit number.
- C adds a zero to the end of a string (It is thus an ASCIIZ string).

Hand in your `changecase.asm` file on `learn.sun.ac.za` before **23:55 on Thursday 13 Augustus**.

3 Bronne

3.1 Wenke - sommeerder

- 'n Integer is 16 bisse, so ons gebruik die ...W funksies (MOVW, ADDW, ens).
- ADDW werk hoofsaaklik net op register AX.
- Registerbank 1, register R0 (ook genoem R8) se adres is 0xFFEF0.

3.2 Wenke - verander kas

Original sentence: `mOET VERKIESLIK TEEN vRYDAG @ 17:00 VOLTOOI WEES![{'ok'}]`
Correct case switch: `Moet verkieslik teen Vrydag @ 17:00 voltooi wees![{'OK'}]`

If we call our assembly function from C using: `verander_kas(sentence);`
then the address where the first character is located in RAM will be sent over to our ASM function. What we need to do:

1. Copy the address of the first character from the stack and store it in a register. Remember that it will be at location SP+4 on the stack.
2. Check if the character at the address we got has a value of 0 (not the character '0')
3. If it is an alphabetical letter, change the case.
4. Increment the address register by 1
5. Repeat the test and change case until we get a 0.
6. If we see it is a 0, we are at the end of the sentance, and we should return to the C code.

3 References

3.1 Hints - adder

- An integer is 16 bits, so we use the ...W asm mnemonics (MOVW, ADDW, etc).
- ADDW werk hoofsaaklik net op register AX.
- Register bank 1, register R0 (aka R8) has an address of 0xFFEF0.

3.2 Hints - change case

Some ASM code that might help in getting started with this, showing how the address of the first character can be obtained:

```
_verander_kas:
  movw HL,SP          ;Stack Pointer na HL
  movw AX,HL          ;Kopiëer na AX
  addw AX,#4          ;Tel 4 by
  movw HL,AX          ;Kopiëer terug na AX
  movw AX,[HL]        ;Gaan haal adres vanaf stapel by SP+4
  movw HL,AX          ;adres van eerste karakter nou in HL
  ; Hierdie kan waarskynlik korter geskryf word
  ; Maak seker watter registers waarvoor gebruik mag word

  ; doen die res van die toetse en bewerkings

  ; as karakter=0, br $klaar

  klaar:
  ret
```

ASM mnemonics you might need to use: `mov, cmp, br, bz, bh, bc, inc`
A branch mnemonic takes a label to jump to: `br $klaar`

3.3 Opsomming van die C-ASM koppelvlak

Dit word ook genoem die *Application binary interface*[2]. Daarom is die *RL78-ABI-Specification.pdf* dokument van toepassing.

Ons beskou:

```
result = function(parameter a, parameter b, ..., parameter x, ..., parameter n)
```

- param a in geheue by adres $SP + 4$
- param b in geheue by adres $SP + 6$
- param x in geheue by adres $SP + 4 + (x \times 2)$
  $x \in \{0, n-1\}$

Die ABI spesifikasies sê dat die inhoud van registerbank 0 en -bank 1 nie tussen funksie-roepe behoue bly nie. Bank 0 en bank 1 se registers kan dus as tydelike stoorplek in jou funksie gebruik word. Al die parameters wat van C af oorgestuur word moet van die stapel gelees word. Integers en karakters word tot 16 bisse opgelyn (daarom maal ons x met 2).

Jou ASM-funksie se resultaat moet in geheue-adres 0xFFEF0 gestoor word sodat ons daarby kan uitkom vanaf C. Dit is register R0 in bank 1.

Onthou dat ons "Little endian" gebruik.

3.3 C-ASM interface summary

The official name for this is an *Application binary interface*. Therefore the *RL78-ABI-Specification.pdf* is applicable here.

Let's look at:

```
result = function(parameter a, parameter b, ..., parameter x, ..., parameter n)
```

- param a in memory at location $SP + 4$
- param b in memory at location $SP + 6$
- param x in memory at location $SP + 4 + (x \times 2)$
  $x \in \{0, n-1\}$

The ABI specifications says that "Registers R0 through R15 (banks 0 and 1) are not preserved across function calls". Bank0 and Bank1 can thus be used as scratch areas. All passed values (parameters) should be read from the stack. Integers and chars are alligned to 16 bits (therefore we multiply x by 2).

The result of your ASM function should be stored at memory address 0xFFEF0 so that we can access it from C. This is register R0 in Bank1.

Remember that we are using little endian!

[2] Adhering to ABIs (which may or may not be officially standardized) is usually the job of the compiler, OS or library writer, but application programmers may have to deal with ABIs directly when writing programs in a mix of programming languages, using foreign function call interfaces between them. - Wikipedia on ABI's

3.4 ASCII table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com