Silesian University of Technology

Department of Graphics, Computer Vision and Digital Systems

Year	Type*: SSI/NSI/NSM	Subject: Assembler Programming Languages	Group	Section
2024/2025	SSI	APL – LAB	1	1
Tutor:	dr inż. Adam Opara		Class date: (week day, hour)	
Section:	1. Piotr Copek 2. Zuzanna Micorek		09/05/2025	
Contact Email:	pc21399@student.polsl.pl		8.30-10.00	
Report				

Task 1

Analyze the operation of `FindChar_1 ... 6`. What errors have been found and how they have been corrected (table in the report)?

Procedure	Error Found	Correction Applied
FindChar_1	Used AL instead of AH for comparison	Changed to consistent AH usage
	Missing RET after Got_Equal label	Added proper return instruction
FindChar_2	Local string defined in code section	Moved to proper data segment
	Incorrect string termination check	Fixed FFh comparison
FindChar_3	Stack frame not properly managed	Added EBP frame setup/ cleanup
FindChar_4	Mixed DataString[ESI] and [ESI] syntax	Standardized to [ESI] access
FindChar_5	Fall-through after match condition	Added explicit RET after Found5
FindChar_6	Wrong jump target (JE Not_Find)	Corrected to JE Got_Equal
ReadTime_1	Missing register preservation	Added push/pop for EBX, ECX

Task 2

Try the ReadTime_1 procedure. What is RDTSC?

- RDTSC (Read Time-Stamp Counter) reads the processor's 64-bit timestamp counter
- Returns result in EDX:EAX (high/low 32 bits respectively)
- Measures CPU clock cycles

Task 3

Are the time measurements repeatable and what it comes from? When is this method of measurement reliable?

Repeatability factors:

- CPU frequency scaling
- Interrupts and context switches
- Cache warm-up effects

Reliable when:

- Single-threaded, pinned CPU frequency
- Minimal background processes
- Multiple runs for statistical significance
- Fixed input data size

Task 4

Why is the timing value with a comma sign, for some instructions?

Notation	Interpretation
`1`	Base execution cycles
`1,1`	Decode + Execute stages
`2,1`	Memory access + Execute

Instruction	Binary	Timing	Fields
`MOV ESI, OFFSET Data`	`BE 00000000`	1	Opcode + Immediate
`CMP [EBX+ESI],'J'`	`80 3C 33 4A`	2,1	ModR/M + SIB + Imm8

- **First number**: Front-end cost (decode/address calc)
- **Second number**: Back-end cost (execution)
- Memory accesses add pipeline stages

Task 5

For the selected instruction (the more complicated the better) write the binary code it has been translated to; specify individual fields of the instruction.

Field	Binary Value	Description
Full Hex	`80 3C 33 4A`	Complete machine code
Prefix	None	No segment override or REX prefix
Opcode	`80 /7 ib`	CMP r/m8 with imm8
ModR/M	`3C`	`[--][--][SIB]` + imm8 operand
Mod	`00`	No displacement
Reg/Op	`111` (/7)	CMP operation code
R/M	`100`	SIB byte follows
SIB	`33`	`[EBX + ESI*1]` addressing
Scale	`00`	×1 scaling
Index	`110`	ESI register
Base	`011`	EBX register
Immediate	`4A`	ASCII 'J' (0x4A)

Task 6

Write your own My_Procedure procedure to search for a character in a string, trying to make its execution time as fast as possible.

```
My_Procedure PROC
    mov ebx, OFFSET DataString
    mov ah, 'J'
    xor eax, eax        ; default: not found

    cmp BYTE PTR [ebx+0], ah
    je FoundM
    cmp BYTE PTR [ebx+1], ah
    je FoundM
    cmp BYTE PTR [ebx+2], ah
    je FoundM
    cmp BYTE PTR [ebx+3], ah
    je FoundM
    cmp BYTE PTR [ebx+4], ah
    je FoundM
    cmp BYTE PTR [ebx+5], ah
    je FoundM
    cmp BYTE PTR [ebx+6], ah
    je FoundM

    ret
FoundM:
    mov eax, 1
    ret
My_Procedure ENDP
```

Original character search:

```
Processing time: 336
Processing time: 219
Processing time: 283
Processing time: 282
Processing time: 280
```

My character search:

```
Processing time: 30
Processing time: 24
Processing time: 23
Processing time: 24
Processing time: 168
```

Task 7

Based on the timing listings, place a table of execution times for all procedures searching the character. Take into account the number of the loop executions and taken/not taken jumps.

Procedure	Min Cycles	Max Cycles	Avg Cycles	Loop Executions	Taken Jumps	Not Taken Jumps
FindChar_1	2482	2482	447.0	4	3	1
FindChar_2	20	197	43.5	4	3	1
FindChar_3	39	42	40.25	4	3	1
FindChar_4	233	323	285.75	4	3	1
FindChar_5	22	272	136.5	4	3	1
FindChar_6	35	358	197.75	4	3	1
My_Procedure	25	168	42.75	4	3	1

Conclusions

The procedures initially had issues like using wrong registers, incorrect memory access, and missing instructions, which were fixed to improve correctness and performance. The `RDTSC` instruction was used to measure execution time in CPU cycles, but its results can vary due to CPU frequency changes, background processes, and cache effects. Instructions involving memory access or complex addressing take more cycles, which was seen in their timing values. The custom procedure performed the fastest because it avoided loops and reduced jumps by unrolling the code. Overall, optimizing the code led to much better performance.