


Politechnika Wrocławska

Technika cyfrowa 2


wykład 6

Katedra Metrologii Elektronicznej i Fotonicznej
Andrzej Stępień



Literatura


- J. Janiczek, A. Stępień: *Mikrokontrolery*. WCKP, Wrocław, 1997
- J. Janiczek, A. Stępień: *Laboratorium systemów mikroprocesorowych. Cz. 1. WEZN, Wrocław, 1995*
- J. Janiczek, A. Stępień: *Laboratorium systemów mikroprocesorowych. Cz. 2. WEZN, Wrocław, 1996*
- T. Starecki: *Mikrokontrolery 8051 w praktyce. BTC, Warszawa 2002*
- MSP430x4xx. *User's Guide. SLAU056C, Texas Instruments, 2003*
- ST7 FAMILY. *PROGRAMMING MANUAL. STMicroelectronics, March 1999*
- ST7. *8-BIT MCU FAMILY. USER GUIDE. STMicroelectronics, July 2002*



Jak rozwiązać problem ?

Jeśli w języku wysokiego poziomu (ALGOL/FORTRAN, rok 1950) zwiększyć jego efektywność i wydajność przez wprowadzenie podprogramów, to jak:

- zapamiętać adres powrotu z podprogramu ?
- przenieść zmienne z programu głównego do tego podprogramu ?
- zapamiętywać wartości chwilowe obliczeń ?
- uprościć obliczenia, np. typu $X = (A + B) * (C + D)$?



Po co komu stos ?

Rozwiązanie

- wprowadzić **stos** (Stack), liniową strukturę danych traktowaną jako **rejestr** typu **LIFO** (Last In First Out)
- wprowadzić instrukcje procesora typu:
 - **CALL addr**, wywołanie podprogramu – odesłanie na stos adresu powrotu z podprogramu
 - **RET**, powrót z podprogramu – pobranie ze stosu adresu powrotu, adresu dalszej części programu głównego
 - **PUSH arg**, odesłanie argumentu na stos
 - **POP arg**, pobranie argumentu ze stosu


Tylko procesory współpracujące ze stosem mogą:

- przyjmować przerwania
- wykonywać podprogramy obsługi przerwań




Typy stosu

- **sprzętowy**:
 - stos złożony z N dodatkowych, równoległych rejestrów lub rejestrów przesuwanych typu LIFO
 - szybkie (w stosunku do realizacji programowej) odwołania do stosu mały rozmiar stosu (w stosunku do realizacji programowej)
- **programowy**:
 - adresowanie dodatkowym rejestrem, wskaźnikiem stosu SP (Stack Pointer)
 - automatyczna inkrementacja / dekrementacja wskaźnika stosu przy każdorazowym odwołaniu do stosu
 - alokacja stosu w pamięci RAM
 - rozmiar stosu ograniczony rozmiarem dostępnej pamięci RAM



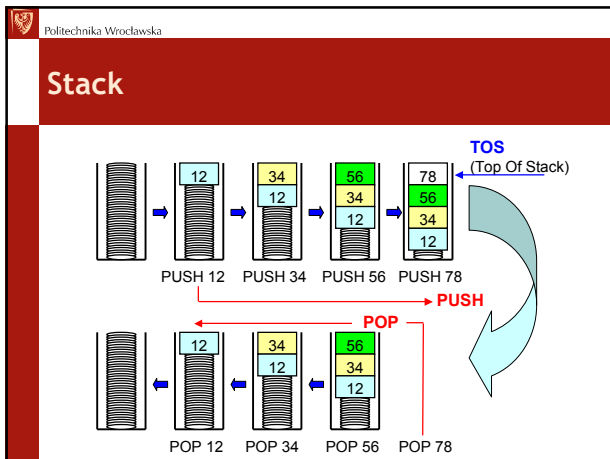
Stack Computers



Philip J. Koopman, Jr.

Originally published by Ellis Horwood in 1989

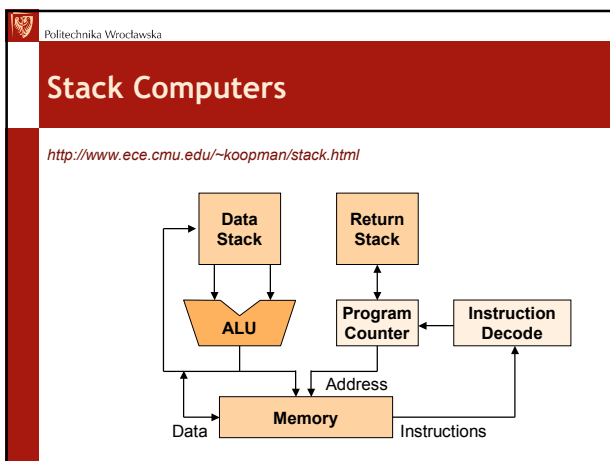
http://www.ece.cmu.edu/~koopman/stack_computers/index.html



Politechnika Wrocławska

Stack Computers - Philip Coopman

- Stacks are simple, a child intuitively understands a stack of things and how it works.
- A stack is a data structure that is accessed from one end or it is a Last In First Out buffer.
- Stack Computers are machines that have hardware and instructions to facilitate the use of stacks in programs. In the hardware within a processor a stack may be implemented using pointers on the processor or in memory to locations in memory. It can also be implemented as on-chip registers or it may have part of the stack cached in on-chip registers and have the rest of the stack in memory.



Politechnika Wrocławska

Single vs. multiple stacks - Philip Coopman (1/2)

- Single Stack** computers are those computers with exactly one stack supported by the instruction set. This stack is often intended for state saving for subroutine calls and interrupts. It may also be used for expression evaluation. In either case, it is probably used for subroutine parameter passing by compilers for some languages.
- An advantage of having a **single stack** is that it is easier for an operating system to manage only one block of variable sized memory per process.
- A disadvantage of a **single stack** is that parameter and return address information are forced to become mutually well nested.

Politechnika Wrocławska

Single vs. multiple stacks - Philip Coopman (1/2)

- Multiple Stack** computers have two or more stacks supported by the instruction set:
 - one stack is usually intended to store return addresses,
 - the other stack is for expression evaluation and/or subroutine parameter passing. The parameter stack is separate from the return address stack.
- Multiple stacks** allow separating control flow information from data operands.
- An important advantage of having multiple stacks is one of speed. **Multiple stacks** allow access to multiple values within a clock cycle. As an example, a machine that has simultaneous access to both a data stack and a return address stack can perform subroutine calls and returns in parallel with data operations.

Politechnika Wrocławska

Stos sprzętowy - PIC17C4X (Microchip Technology)

- stosem jest 16-bitowy **bufor pierścieniowy**; stos nie jest fragmentem pamięci programu lub danych, nie jest rejestrem specjalnym
- bufor stosu jest **16-poziomowy**
- 17-a operacja 'PUSH' **nadpisuje wynik** pierwszej operacji 'PUSH', 18-a operacja - drugiej itd.
- wskaźnik stosu jest **zerowany** po wszystkich typach zerowania procesora
- wskaźnik stosu **nie jest dostępny** do odczytu lub zapisu
- operacja **'PUSH'** jest wywołanie podprogramu lub przyjęcie przerwania (*brak instrukcji PUSH*)
- operacja **'POP'** jest powrót z podprogramu lub obsługi przerwania (*brak instrukcji POP*)
- jedynym znacznikiem stanu wskaźnika stosu jest **STKAV** (*STack AVailable bit*):
 - STKAV = 1 po zerowaniu procesora,
 - STKAV = 0 po osiągnięciu przez wskaźnik stosu maksymalnej wartości 0Fh

Politechnika Wrocławska

Stack Pointer (SP) (1/2)

Bottom of the stack: 1FFh = Reset Value (ST72334) → Stack Higher Addr (256 bytes (ST72334))

Top of the stack: 100h → Stack Lower Addr (128 bytes (ST72264))

Fixed By HW

- The stack pointer is a 16-bit register. The 8 least significant bits contain the address of the next free location of the stack.

0 0 0 0 0 0 0 1 x x x x x x x x

Fixed By HW

Politechnika Wrocławska

Stack Pointer (SP) (2/2)

Stack Pointer (SP):

0 0 0 0 0 0 0 1 x x x x x x x x

- Since the stack is 128 or 256 bytes deep, the 8th most significant bits are forced by hardware.
- Following an MCU Reset, or after a Reset Stack Pointer instruction (RSP), the Stack Pointer contains its reset value (=1FFh for ST723xx or =17Fh for ST722xx) which is the stack higher address.

Note: When the lower limit is exceeded, the Stack Pointer wraps around to the stack upper limit, without indicating the stack overflow. The previously stored information is then overwritten and therefore lost. The stack also wraps in case of an underflow. (**Stack overflow is not indicated**)

Politechnika Wrocławska

Stack Manipulation

SP=1FFh

100h

After Reset: SP=1FFh

CALL Subroutine: PCH, PCL pushed

PUSH Y: Y pushed

POP A: A popped

RET RSP: PCH, PCL popped, SP restored

Politechnika Wrocławska

PUSH - push into the Stack

Syntax PUSH src e.g.: push A
src: A, X, Y, CC

Operation (SP--) <= dst

Description
Save into the stack the dst byte location. The stack pointer is decremented by one. Used to save a register value.

Politechnika Wrocławska

POP - pop from Stack

Syntax POP dst e.g.: pop CC
dst: A, X, Y, CC

Operation dst <= (++SP)

Description
Restore from the stack a data byte which will be placed in dst location. The stack pointer is incremented by one. Use to restore a register value.

Politechnika Wrocławska

CALL - wywołanie podprogramu

CALL CALL Subroutine (Absolute)
CALLR CALL Subroutine Relative

Syntax CALL dst e.g.: call divide32_16
CALLR dst e.g.: callr divide32_16

Operation
PC = PC + lgth
(SP--) = LSB (PC)
(SP--) = MSB (PC)
PC = dst

Description
The current PC register value is pushed onto the stack, then PC is loaded with the (*relative*) destination address. This instruction should be used versus CALLR when developing a program.

Politechnika Wrocławska

CALL - CALL subroutine

	CALL Subroutine (Absolute)	CALLR Subroutine Relative
Syntax	CALL dst <i>CALLR dst</i>	e.g.: call divide32_16 <i>e.g.: callr divide32_16</i>
Operation	PC = PC + lgth (SP--) = LSB (PC) (SP--) = MSB (PC) PC = dst	
Description	The current PC register value is pushed onto the stack, then PC is loaded with the (<i>relative</i>) destination address. This instruction should be used versus CALLR when developing a program.	

Politechnika Wrocławska

RET - return from subroutine

Syntax	RET
Operation	MSB (PC) = (++SP) LSB (PC) = (++SP)
Description	Restore the PC from the stack. The stack pointer is incremented twice. This instruction is the last one of a subroutine.

Politechnika Wrocławska

RSP - reset Stack Pointer

Syntax	RSP
Operation	SP = Reset Value (=1FFh)
Description	Reset the stack pointer to its reset initial value. This instruction may be put as first executed instruction in the reset routine.
Trick	It may be used to test current stack size used with an ST7 independent program.

Politechnika Wrocławska

Stack Pointer (SP)

- wydzielony logicznie obszar wewnętrznej pamięci RAM (IDATA)
- adresowany wskaźnikiem stosu SP (Stack Pointer)
- po sprzętowym zerowaniu procesora SP = 7 (adres R0 w banku RB1)

segment bitowy; 128 bitów: 0 .. 7Fh
4 banki rejestrów: R0 .. R7

Politechnika Wrocławska

Stack Manipulation

SP=07h
After Reset
CALL Subroutine
PUSH B
POP Acc
RET

Politechnika Wrocławska

PUSH - push into the Stack / POP - pop from Stack

Syntax	PUSH addr
Operation	(++SP) ← dst ; SP ← SP + 1 ; (SP) ← (addr)
Syntax	POP addr
Operation	dst ← (SP--) ; (addr) ← (SP) ; SP ← SP - 1 addr, e.g.: only Acc not A

Politechnika Wrocławska		
CALL - CALL subroutine / RET - return from subroutine		
ACALL addr_11	; PC ← PC + 2 ; INC SP ; (SP) ← PC _{7..0} ; INC SP ; (SP) ← PC _{15..8} ; PC ← addr _{10..0}	2 bajty, 2 cykle maszynowe
LCALL addr_16	; PC ← PC + 3 ; INC SP ; (SP) ← PC _{7..0} ; INC SP ; (SP) ← PC _{15..8} ; PC ← addr _{15..0}	3 bajty, 2 cykle maszynowe
RET	; PC _{15..8} ← (SP) ; DEC SP ; PC _{7..0} ← (SP) ; DEC SP	1 bajt, 2 cykle maszynowe

Politechnika Wrocławska		
Deklaracja stosu		
ISEG	AT	30h ; deklaracja adresu początku segmentu ; w wewnętrznej pamięci danych IDATA ; rezerwacja 8 bajtów
Stos:	DS	8
CSEG	AT	0 ; deklaracja adresu początku segmentu ; w pamięci programu CODE
Początek:		
MOV	SP, #Stos-1	; inicjalizacja stosu użytkownika ; dalsza część programu użytkownika
Programy uruchomieniowe (debugger) w trakcie:		
• pracy krokowej		
• pracy z pułapkami		
wykorzystują stos do zapamiętywania wartości chwilowych niektórych rejestrów uruchamianego programu (2 .. 8 bajtów).		

Politechnika Wrocławska		
Przykład dostępu do stosu		
przesunąć zawartość akumulatora do rejestru R4 w banku RB1 i do rejestru B poprzez stos założony w wewnętrznej pamięci RAM od adresu 30h.		
Wymiana:		
MOV	SP, #30h-1	; SP ← 30h - 1
PUSH	Acc	; SP ← SP + 1
		; (SP) ← (Acc)
PUSH	Acc	; powtórzenie wpisu do stosu
POP	8 + 4	; (8+4) ← (SP), 8+4 to adres R4 w RB1
POP	B	; powtórzenie odczytu ze stosu
		; dalsza część programu

Politechnika Wrocławska		
Ochrona stanu rejestrów w podprogramach		
jeśli nie zawierają one parametrów wejściowych lub wyjściowych do lub z podprogramów:		
CALL	Subroutine	; wywołanie podprogramu
Subroutine:		
PUSH	Acc	; ochrona stanu akumulatora A
PUSH	PSW	; ochrona stanu rejestru statusowego PSW
PUSH	DPL	; ochrona stanu wskaźnikowego rejestru DPTR _{LOW}
PUSH	DPH	; ochrona stanu wskaźnikowego rejestru DPTR _{HIGH}
		; właściwy podprogram
POP	DPH	; odtworzenie stanu wskaźnikowego rejestru DPTR _{HIGH}
POP	DPL	; odtworzenie stanu wskaźnikowego rejestru DPTR _{LOW}
POP	PSW	; odtworzenie stanu rejestru statusowego PSW
POP	Acc	; odtworzenie stanu akumulatora A
RET		; zakończenie podprogramu

Politechnika Wrocławska		
Stos i podprogramy w x86		
Przetwarzanie danych – wnoszenie argumentów do podprogramów:		
▪ dostęp do szczytu stosu (TOS) – adresowanie stosu tylko wskaźnikiem stosu SP		
▪ dostęp do dowolnego elementu stosu – adresowanie stosu wskaźnikiem stosu SP + offset (przesunięcie):		
Turbo Pascal (Intel x86):		
<div> <div> <div>Auto-inkrementacja SP</div> <div>Auto-dekrementacja SP</div> </div> <div> <div>TOS</div> <div>TOS</div> </div> <div> <div>12</div> <div>34</div> <div>56</div> <div>78</div> <div>90</div> <div>01</div> </div> <div> <div>Arg1 → DX</div> <div>Arg2 → AX</div> <div>adres powrotu z podprogramu</div> </div> </div>		
<div> <div>Procedure ProcU (Arg1, Arg2: Integer);</div> <div>ProcU</div> <div>MOV BX, SP</div> <div>MOV AX, SS:[BX+2]</div> <div>MOV DX, SS:[BX+4]</div> <div>RET 4</div> </div>		

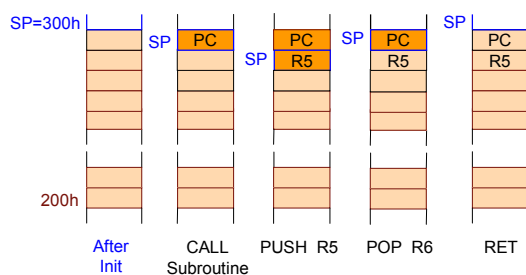
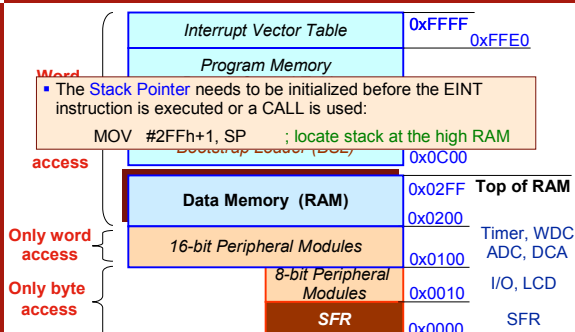
Politechnika Wrocławska		
Stack		
The system stack pointer (SP) is used for the storage of the following items:		
<ul style="list-style-type: none"> Interrupt return addresses and status register contents Subroutine return addresses Intermediate results Variables for subroutines, floating point package etc. 		
The system Stack Pointer SP is always decremented / incremented by two, independent of the byte suffix		
The Stack Pointer always points to even addresses. This means the LSB is always zero. Odd SP addresses will end up in non-predictable results.		
If bytes are pushed on the system stack, only the lower byte is used, the upper byte is not modified:		
PUSH	#05h	; 0005h → TOS
PUSH.B	#05h	; xx05h → TOS

- | | |
|----|-----|
| 15 | 1 0 |
| | 0 |

- | | |
|----|-----|
| 15 | 1 0 |
| | 0 |

- Initialize the SP, typically to the top of RAM:

- *R4 .. R15 - user working register*



- The system Stack Pointer SP is also used by the interrupt routine service

PUSH	SR	; save status register
PUSH.B	R8	; save lower bytes of R8 (R8 _{7:0})

- The system Stack Pointer SP is also used by the RET and RETI instructions

- The branch instruction is a word instruction

- (SP) \Rightarrow PC
SP + 2 \Rightarrow SP

- Status, OscOff, CPUOff and GIE bits are not affected

```
RET          ; return from subroutine
; core instruction: MOV @SP+, PC
```

Example: R4 is to be used as a software stack pointer.

WORD Stack

```
MOV    item, 0(R4)      ; Push item on stack
```

```
MOV    @R4+, item2      ; Pop item from stack
```

BYTE Stack

MOV.B	item, 0(R4)	; Push item on stack
-------	-------------	----------------------

MOV.B @R4+, item2 ; Pop item from stack

The diagram illustrates the return sequence from a subroutine. It shows the stack state before and after the RET instruction.

Before the CALL: The stack contains three entries: Argument0, Argument1, and Return Address. The stack pointer (SP) points to the Return Address.

After the CALL: The stack contains three entries: Return Address, Return Address, and Return Address. The stack pointer (SP) points to the top Return Address.

After the RET: The stack contains three entries: Return Address, Return Address, and Return Address. The stack pointer (SP) points to the top Return Address.

User	usr	Normal program execution mode; unprivileged mode under which most tasks run
System	sys	Runs privileged operating system task; privileged mode using the same registers as user mode
Supervisor	svc	A protected mode for the operating system; entered on reset and when a Software Interrupt instruction is executed
Abort	abt	Implements virtual memory and/or memory protection; used to handle memory access violations
Undefined	und	Supports software emulation of hardware coprocessors; used to handle undefined instructions
IRQ	irq	Used for general-purpose interrupt handling; entered when a low priority (normal) interrupt is raised
FIQ	fiq	Supports a high-speed data transfer or channel process; entered when a high priority (fast) interrupt is raised

Mode changes: software control, external interrupts or exception processing.

User	System	Privileged Modes		Exception Modes			Fast
		Supervisor	Abort	Undefined	Interrupt	Interrupt	
R0	R0	R0	R0	R0	R0	R0	
R1	R1	R1	R1	R1	R1	R1	
R2	R2	R2	R2	R2	R2	R2	
R3	R3	R3	R3	R3	R3	R3	
R4	R4	R4	R4	R4	R4	R4	
R5	R5	R5	R5	R5	R5	R5	
R6	R6	R6	R6	R6	R6	R6	
R7	R7	R7	R7	R7	R7	R7	
R8	R8	R8	R8	R8	R8	R8 <i>fix</i>	
R9	R9	R9	R9	R9	R9	R9 <i>fix</i>	
R10	R10	R10	R10	R10	R10	R10 <i>fix</i>	
R11	R11	R11	R11	R11	R11	R11 <i>fix</i>	
R12	R12	R12	R12	R12	R12	R12 <i>fix</i>	
R13	R13	R13 <i>svc</i>	R13 <i>abt</i>	R13 <i>und</i>	R13 <i>irq</i>	R13 <i>fix</i>	
R14	R14	R14 <i>svc</i>	R14 <i>abt</i>	R14 <i>und</i>	R14 <i>irq</i>	R14 <i>fix</i>	
PC	PC	PC	PC	PC	PC	PC	

CPSR CPSR CPSR mod CPSR mod CPSR mod CPSR mod CPSR mod

SPSR SPSR SPSR SPSR SPSR SPSR SPSR

The diagram illustrates the relationship between registers in different ARM modes. On the left, under 'User & System', is the **R13** register, which is also labeled **PC** (Program Counter). Below it is the **CPSR** (Current Program Status Register). On the right, under 'Other Modes', is the **R13_mod** register, which is also labeled **R14_mod PC**. Below it is the **CPSR mod** register. A double-headed arrow connects the **R13** register to the **R13_mod** register, indicating that they share the same physical register but are mapped to different addresses depending on the current mode.

Politechnika Wrocławska

Register R14

In each mode register **R14** is used to **hold subroutine return address**. The subroutine return is performed by copying R14 back to the program counter PC.

User & System: R13, R14, PC, CPSR

Other Modes: R13_mod, R14_mod, PC, CPSR_mod, SPSR_mod

Politechnika Wrocławska

Kod ASCII

Kod ASCII (American Standard Code for Information Interchange):

- 7-bitowy kod konwersji cyfr, liter, znaków interpunkcyjnych i kodów sterujących;
- 8-bitowy rozszerzony kod ASCII zawierający znaki narodowe

znak	ASCII	znak	ASCII	znak	ASCII
LF	10 0Ah	@	64 40h	`	96 60h
CR	13 0Dh	A	65 41h	a	97 61h
space	32 20h	F	70 46h	f	102 66h
!	33 21h	G	71 47h	g	103 67h
/	47 2Fh	Y	89 59h	y	121 79h
0	48 30h	Z	90 5Ah	z	122 7Ah
1	49 31h	[91 5Bh	{	123 7Bh
9	57 39h	^	94 5Eh	~	126 7Eh
:	58 3Ah	_	95 5Fh	DEL	127 7Fh

Arrows between the middle tables indicate a shift of +20h and -20h between the standard and extended ASCII sets.

Politechnika Wrocławska

Wskaźnik 7-segmentowy (1/2)

Przedstawić zawartość komórki pamięci o adresie addr, w postaci dwóch znaków w kodzie 7-segmentowym. Wynik zapisać w komórkach pamięci o adresach:

- addr_low - część bardziej znacząca

SEGMENT IDENTIFICATION

NUMERICAL DESIGNATIONS AND RESULTANT DISPLAYS

SN7447. BCD-TO-SEVEN-SEGMENT DECODERS/DRIVERS

Politechnika Wrocławska

Wskaźnik 7-segmentowy (2/2)

```

MOV A, addr      ; A ← (addr)
SWAP A           ; A7..4 ⇌ A3..0
CALL Hex_7_Segm ; wywołanie podprogramu zamiany
MOV addr_high, A ; (addr_high) ← A, przesłanie wyniku części MSB

MOV A, addr      ; A ← (addr)
CALL Hex_7_Segm ; wywołanie podprogramu zamiany
MOV addr_low, A  ; (addr_low) ← A, przesłanie wyniku części MSB

```

Hex_7_Segm: podprogram zamiany

```

ANL A, #0Fh      ; wydzielenie 4 mniej znaczących bitów
MOV DPTR, #Tablica; DPTR ← Tablica
MOVC A, @A+DPTR  ; A ← (A+DPTR)CODE
RET              ; koniec podprogramu

```

Tablica:

```

DB 3Fh, 06h, 5Bh, 4Fh, 66h, 6Dh, 7Dh, 07h, 7Fh, 6Fh ; kody 7-segmentowe cyfr: 0..9
DB 77h, 7Ch, 39h, 5Eh, 79h, 71h ; kody 7-segmentowe liter: A..F

```