ZX Spectrum IDE: Part #3 — A Brief Overview of the Z80 CPU

DATE: JANUARY 25, 2018 POSTED BY: ISTVAN NOVAK COMMENTS: 1 COMMENT
CATEGORY: ZX SPECTRUM TAG: SPECTNETIDE

As I mentioned in the previous post, the soul of the ZX Spectrum 48K microcomputer—what a surprise—is the Z80 CPU. Obviously, you need to emulate the CPU to get closer to a full Spectrum emulator. Believe it or not, CPU emulation is not a big challenge compared to other devices of the machine (video display generation, tape emulation, etc.), but it is laborious due to the richness of Z80's instruction set.

Before going into the implementation details I used in SpectNetIde, I give you a brief overview of Z80A. Please, do not expect this article as a tutorial to learn about the CPU. I will focus on the aspects that are the most essential from the emulator design and development point of view.

The manufacturer of Z80A, Zilog, is an American manufacturer of microcontrollers. Its most known product is the Z80 family of chips. According to its simplicity from hardware interfacing point of view, it became popular right after its initial issue in 1976.

The Z80A version—the one used in ZX Spectrum 48K—is an improved model that increased the maximum speed of the original Z80 from 2.5 MHz to 4 MHz. The chip is an 8-bit CPU with 8-bit and 16-bit registers and provides over 1000 instructions.

Registers

As **Figure 1** depicts, it has 8 main registers (**A**, **B**, **C**, **D**, **E**, **H**, **L**, and **F**) with their corresponding alternate registers (**A'**, **B'**, **C'**, **D'**, **E'**, **H'**, **L'**, and **F'**); two index registers (**IX**, **IY**); a stack pointer (**SP**); a program counter (**PC**), and two special registers, **I** and **R**.

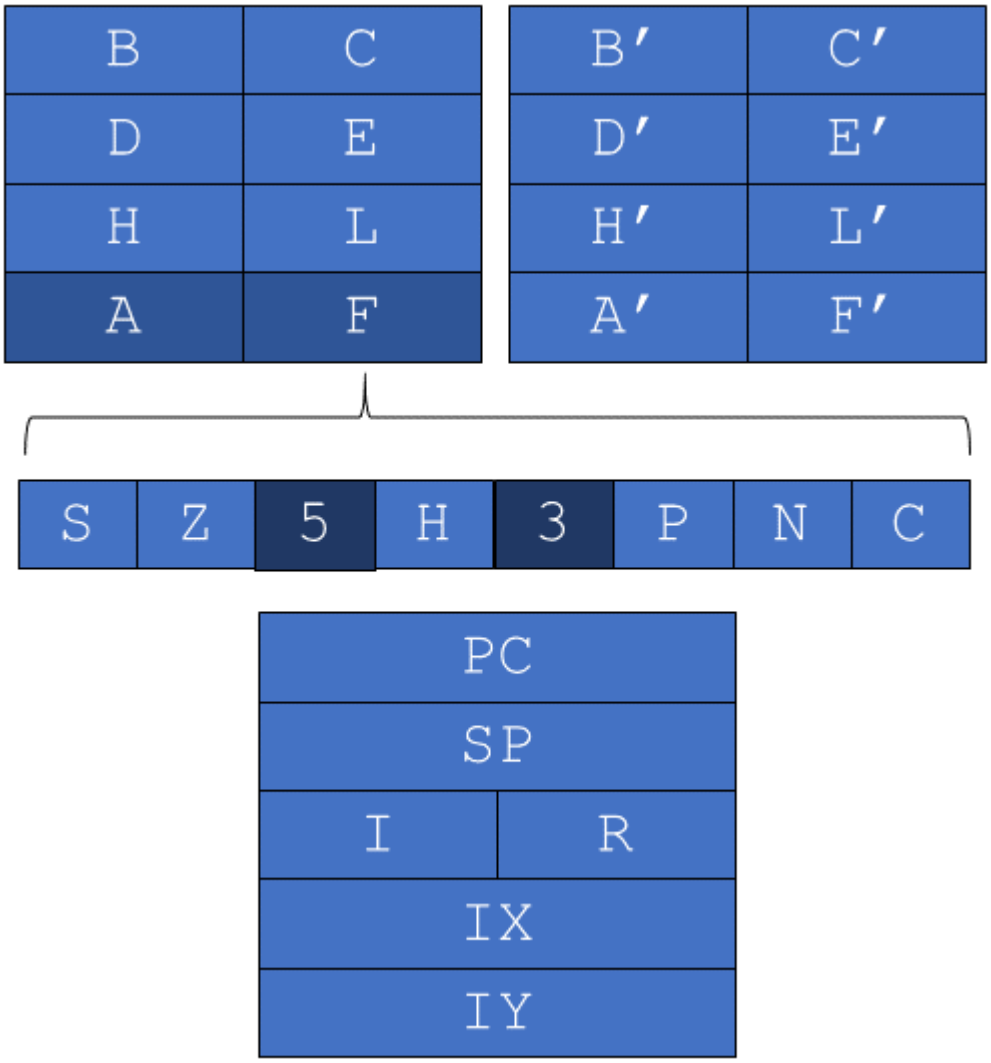

Figure 1: The registers of the Z80A CPU

The **A** register is called Accumulator. From the programming point of view, this register is the most flexible, as it can be the operand of more instructions than any others. The **F** register is a set of 1-bit flags that are individually set by Z80 instructions. The CPU can use the state of flags as conditions for instructions that change the program flow.

The main 8-bit registers can be used in 16-bit pairs. Thus **B** and **C** compose **BC** (the 8 bits of **B** are the upper, the bits of **C** the lower parts of the resulting 16-bit register). Similarly, you have **DE** and **HL**.

The alternate register set can serve as a backup for the main registers. A Z80 instruction, **EXX**, swaps the main and alternate pairs.

The **PC** (Program Counter) points to the location the CPU should read the next byte of the operation code to execute. **SP** stands for Stack Pointer—as its name suggests, it points to the top of the stack used for subroutine calls and stack-saved values.

IX and **IY** are index registers. You can use their contents as a memory pointer in tandem with an 8-bit displacement and read or write the composed address. For example, if **IX** contains **$4000**, an instruction reading the **(IX+$2A)** address—here, **$2A** is the displacement—will obtain the contents of address **$402A**.

There are two special registers, **I** and **R**. **I** (Interrupt Page Address register) is used in a particular interrupt mode as the high-order eight bits of the memory location that serves as the interrupt routine address—the device requesting the interrupt provides the low-order eight bits. **R** (Memory Refresh register) contains a memory refresh counter enabling dynamic memories to be used with the same ease as static memories.

Control Signals

The CPU uses control signals to communicate with external devices. Vice-versa, it receives signals when the external world wants to notify the CPU about events. The Z80 has thirteen control pins, 8 of them send, five

of them gets signals. From the emulator points of view, these three signals arriving at the CPU are essential:

- **`INT`**: interrupt request from devices. This request can be disabled (enabled) by with CPU instructions.
- **`NMI`**: Non-maskable interrupt request (cannot be enabled or disabled)
- **`RESET`**: Resets the CPU immediately just as if we turned the power off and then on again.

Instructions

The Z80 CPU has more than 1000 instructions. The most often used ones have a single-byte operation code. There are less frequently used instructions that use two or three bytes of operation codes. Besides these, instructions may have arguments. Each instruction starts with the operation code. These codes explicitly tell the CPU whether the instruction has arguments, or its code has multiple bytes.

- The **`$ED`** operation code is a prefix that means a second byte specifies the extended operation.
- **`$CB`** is another prefix code. In this case the second byte names a bit manipulation instruction.
- **`$DD`** and **`$FD`** are prefixes for indexed operations. A second byte describes the instruction. **`$DD`** means that the **`IX`**, **`$FD`** implies that the **`IY`** index register should be used in the instruction.

- When a **$CB** prefix follows the **$DD** and **$FD** prefixes, a third byte names the bit manipulation instruction.
- When writing Z80 code, we use the Z80 Assembler language to define the instructions to execute. The compiler translates these very instructions to their machine code equivalent, to operation codes and arguments, respectively.

Let's see a few examples:

- The **INC BC** instruction has a single operation code, **$03**. *(This instruction increments the 16-value of the **BC** register pair with one.)*
- The **LD A,$4C** instruction is a standard operation (code **$3C**) and has an argument, **$4C**. The entire operation is these two bytes, in this order: **$3E**, **$4C**. *(This instruction loads the 8-bit value **$4C** into the Accumulator.)*
- The **LD ($4238),A** instruction is a standard operation (code **$32**) and has a 16-bit argument of **$4238**. The argument follows the operation code in LSB/MSB order (the least significant byte then the most significant one), so the entire operation contains these three bytes: **$32**, **$38**, **$42**. *(Stores the value of **A** in the **$4238**)*
- The **NEG** instruction is an extended operation with the **$ED** prefix followed by the **$44** operation code, so it consists these two bytes: **$ED**, **$44**. *(Calculates the two's complement of **A**.)*
- The **BIT 0,E** instruction tests the leftmost bit of the **E** register and sets status flags accordingly. It's a bit manipulation operation (**$CB**

prefix) with the operation code of **$43**. So, the entire operation is composed of these bytes: **$CB**, **$43**.

- The **LD (IX+$3C),$87** operation stores **$87** in the memory address calculated from the current value of **IX** plus the **$3C** displacement. It starts with the **$DD** prefix and the **$36** operation code. The entire operation contains the **$3C** displacement, and then **$87** argument. So, altogether it has four bytes: **$DD**, **$36**, **$3C**, and **$87**.
- The **RLC (IY+$2F),C** instruction starts with the **$FD** prefix (it is **IY**-indexed), then goes on with the **$CB** prefix (bit manipulation). There are two more bytes, **$01**, the operation code, and **$2F**, the displacement, respectively. The entire operation has these four bytes: **$FD**, **$CB**, **$01**, **$2F**.

> **Note**: In this article, I do not intend to teach you Z80 instructions in details. If you're interested in Z80 Assembler programming, you'll find enough information. Here are two pages to start:
> http://sgate.emt.bme.hu/patai/publications/z80guide/
> http://z80-heaven.wikidot.com/system:tutorials

Undocumented Instructions and Registers

Th official Z80 documentation—I do not know why—omits hundreds of operations the Z80 can execute. Many of these are related to the higher and lower eight bits of the **IX** and **IY** index registers (named **XL**, **XH**, **YL**, and **YH**; or sometimes **IXL**, **IXH**, **IYL**, and **IYH**).

> **Note**: Fortunately, you can find reliable documents on the internet, which give you those missing details. You need to know that many ZX Spectrum games utilize these undocumented instructions, so a high-fidelity emulator must implement them—this it what SpectNetIde does, too.

Figure 2, **3**, **4**, **5**, and **6** show the entire instruction set of Z80. The reddish cells are the initially undocumented instructions of the CPU. Please note, **Figure** 5 and **Figure 6** display the **`IX`**-indexed instructions. You can use the same instructions with the **`$FD`** prefix for the **`IY`** register.

Main instructions

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	nop	ld bc,**	ld (bc),a	inc bc	inc b	dec b	ld b,*	rlca	ex af,af'	add hl,bc	ld a, (bc)	dec bc	inc c	dec c	ld c,*	rrca
1	djnz *	ld de,**	ld (de),a	inc de	inc d	dec d	ld d,*	rla	jr *	add hl,de	ld a, (de)	dec de	inc e	dec e	ld e,*	rra
2	jr nz,*	ld hl,**	ld (**),hl	inc hl	inc h	dec h	ld h,*	daa	jr z,*	add hl,hl	ld hl, (**)	dec hl	inc l	dec l	ld l,*	cpl
3	jr nc,*	ld sp,**	ld (**),a	inc sp	inc (hl)	dec (hl)	ld (hl),*	scf	jr c,*	add hl,sp	ld a, (**)	dec sp	inc a	dec a	ld a,*	ccf
4	ld b,b	ld b,c	ld b,d	ld b,e	ld b,h	ld b,l	ld b, (hl)	ld b,a	ld c,b	ld c,c	ld c,d	ld c,e	ld c,h	ld c,l	ld c, (hl)	ld c,a
5	ld d,b	ld d,c	ld d,d	ld d,e	ld d,h	ld d,l	ld d, (hl)	ld d,a	ld e,b	ld e,c	ld e,d	ld e,e	ld e,h	ld e,l	ld e, (hl)	ld e,a
6	ld h,b	ld h,c	ld h,d	ld h,e	ld h,h	ld h,l	ld h, (hl)	ld h,a	ld l,b	ld l,c	ld l,d	ld l,e	ld l,h	ld l,l	ld l, (hl)	ld l,a
7	ld (hl),b	ld (hl),c	ld (hl),d	ld (hl),e	ld (hl),h	ld (hl),l	halt	ld (hl),a	ld a,b	ld a,c	ld a,d	ld a,e	ld a,h	ld a,l	ld a, (hl)	ld a,a
8	add a,b	add a,c	add a,d	add a,e	add a,h	add a,l	add a, (hl)	add a,a	adc a,b	adc a,c	adc a,d	adc a,e	adc a,h	adc a,l	adc a, (hl)	adc a,a
9	sub b	sub c	sub d	sub e	sub h	sub l	sub (hl)	sub a	sbc a,b	sbc a,c	sbc a,d	sbc a,e	sbc a,h	sbc a,l	sbc a, (hl)	sbc a,a
A	and b	and c	and d	and e	and h	and l	and (hl)	and a	xor b	xor c	xor d	xor e	xor h	xor l	xor (hl)	xor a
B	or b	or c	or d	or e	or h	or l	or (hl)	or a	cp b	cp c	cp d	cp e	cp h	cp l	cp (hl)	cp a
C	ret nz	pop bc	jp nz,**	jp **	call nz,**	push bc	add a,*	rst 00h	ret z	ret	jp z,**	BITS	call z,**	call **	adc a,*	rst 08h
D	ret nc	pop de	jp nc,**	out (*),a	call nc,**	push de	sub *	rst 10h	ret c	exx	jp c,**	in a, (*)	call c,**	IX	sbc a,*	rst 18h
E	ret po	pop hl	jp po,**	ex (sp),hl	call po,**	push hl	and *	rst 20h	ret pe	jp (hl)	jp pe,**	ex de,hl	call pe,**	EXTD	xor *	rst 28h
F	ret p	pop af	jp p,**	di	call p,**	push af	or *	rst 30h	ret m	ld sp,hl	jp m,**	ei	call m,**	IY	cp *	rst 38h

Figure 2: Z80 standard instruction set (source: clrhome.org)

Extended instructions (ED)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
4	in b, (c)	out (c),b	sbc hl,bc	ld (**),bc	neg	retn	im 0	ld i,a	in c, (c)	out (c),c	adc hl,bc	ld bc, (**)	neg	reti	im 0/1	ld r,a
5	in d, (c)	out (c),d	sbc hl,de	ld (**),de	neg	retn	im 1	ld a,i	in e, (c)	out (c),e	adc hl,de	ld de, (**)	neg	retn	im 2	ld a,r
6	in h, (c)	out (c),h	sbc hl,hl	ld (**),hl	neg	retn	im 0	rrd	in l, (c)	out (c),l	adc hl,hl	ld hl, (**)	neg	retn	im 0/1	rld
7	in (c)	out (c),0	sbc hl,sp	ld (**),sp	neg	retn	im 1		in a, (c)	out (c),a	adc hl,sp	ld sp, (**)	neg	retn	im 2	
A	ldi	cpi	ini	outi					ldd	cpd	ind	outd				
B	ldir	cpir	inir	otir					lddr	cpdr	indr	otdr				

Figure 3: Z80 extended instruction set (source: clrhome.org)

Bit instructions (CB)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	rlc b	rlc c	rlc d	rlc e	rlc h	rlc l	rlc (hl)	rlc a	rrc b	rrc c	rrc d	rrc e	rrc h	rrc l	rrc (hl)	rrc a
1	rl b	rl c	rl d	rl e	rl h	rl l	rl (hl)	rl a	rr b	rr c	rr d	rr e	rr h	rr l	rr (hl)	rr a
2	sla b	sla c	sla d	sla e	sla h	sla l	sla (hl)	sla a	sra b	sra c	sra d	sra e	sra h	sra l	sra (hl)	sra a
3	sll b	sll c	sll d	sll e	sll h	sll l	sll (hl)	sll a	srl b	srl c	srl d	srl e	srl h	srl l	srl (hl)	srl a
4	bit 0,b	bit 0,c	bit 0,d	bit 0,e	bit 0,h	bit 0,l	bit 0, (hl)	bit 0,a	bit 1,b	bit 1,c	bit 1,d	bit 1,e	bit 1,h	bit 1,l	bit 1, (hl)	bit 1,a
5	bit 2,b	bit 2,c	bit 2,d	bit 2,e	bit 2,h	bit 2,l	bit 2, (hl)	bit 2,a	bit 3,b	bit 3,c	bit 3,d	bit 3,e	bit 3,h	bit 3,l	bit 3, (hl)	bit 3,a
6	bit 4,b	bit 4,c	bit 4,d	bit 4,e	bit 4,h	bit 4,l	bit 4, (hl)	bit 4,a	bit 5,b	bit 5,c	bit 5,d	bit 5,e	bit 5,h	bit 5,l	bit 5, (hl)	bit 5,a
7	bit 6,b	bit 6,c	bit 6,d	bit 6,e	bit 6,h	bit 6,l	bit 6, (hl)	bit 6,a	bit 7,b	bit 7,c	bit 7,d	bit 7,e	bit 7,h	bit 7,l	bit 7, (hl)	bit 7,a
8	res 0,b	res 0,c	res 0,d	res 0,e	res 0,h	res 0,l	res 0, (hl)	res 0,a	res 1,b	res 1,c	res 1,d	res 1,e	res 1,h	res 1,l	res 1, (hl)	res 1,a
9	res 2,b	res 2,c	res 2,d	res 2,e	res 2,h	res 2,l	res 2, (hl)	res 2,a	res 3,b	res 3,c	res 3,d	res 3,e	res 3,h	res 3,l	res 3, (hl)	res 3,a
A	res 4,b	res 4,c	res 4,d	res 4,e	res 4,h	res 4,l	res 4, (hl)	res 4,a	res 5,b	res 5,c	res 5,d	res 5,e	res 5,h	res 5,l	res 5, (hl)	res 5,a
B	res 6,b	res 6,c	res 6,d	res 6,e	res 6,h	res 6,l	res 6, (hl)	res 6,a	res 7,b	res 7,c	res 7,d	res 7,e	res 7,h	res 7,l	res 7, (hl)	res 7,a
C	set 0,b	set 0,c	set 0,d	set 0,e	set 0,h	set 0,l	set 0, (hl)	set 0,a	set 1,b	set 1,c	set 1,d	set 1,e	set 1,h	set 1,l	set 1, (hl)	set 1,a
D	set 2,b	set 2,c	set 2,d	set 2,e	set 2,h	set 2,l	set 2, (hl)	set 2,a	set 3,b	set 3,c	set 3,d	set 3,e	set 3,h	set 3,l	set 3, (hl)	set 3,a
E	set 4,b	set 4,c	set 4,d	set 4,e	set 4,h	set 4,l	set 4, (hl)	set 4,a	set 5,b	set 5,c	set 5,d	set 5,e	set 5,h	set 5,l	set 5, (hl)	set 5,a
F	set 6,b	set 6,c	set 6,d	set 6,e	set 6,h	set 6,l	set 6, (hl)	set 6,a	set 7,b	set 7,c	set 7,d	set 7,e	set 7,h	set 7,l	set 7, (hl)	set 7,a

Figure 4: Z80 bit manipulation instruction (source: clrhome.org)

IX instructions (DD)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0										add ix,bc						
1										add ix,de						
2		ld ix,**	ld (**),ix	inc ix	inc ixh	dec ixh	ld ixh,*			add ix,ix	ld ix,(**)	dec ix	inc ixl	dec ixl	ld ixl,*	
3					inc (ix+*)	dec (ix+*)	ld (ix+*),*			add ix,sp						
4					ld b,ixh	ld b,ixl	ld b,(ix+*)						ld c,ixh	ld c,ixl	ld c,(ix+*)	
5					ld d,ixh	ld d,ixl	ld d,(ix+*)						ld e,ixh	ld e,ixl	ld e,(ix+*)	
6	ld ixh,b	ld ixh,c	ld ixh,d	ld ixh,e	ld ixh,ixh	ld ixh,ixl	ld h,(ix+*)	ld ixh,a	ld ixl,b	ld ixl,c	ld ixl,d	ld ixl,e	ld ixl,ixh	ld ixl,ixl	ld l,(ix+*)	ld ixl,a
7	ld (ix+*),b	ld (ix+*),c	ld (ix+*),d	ld (ix+*),e	ld (ix+*),h	ld (ix+*),l		ld (ix+*),a					ld a,ixh	ld a,ixl	ld a,(ix+*)	
8					add a,ixh	add a,ixl	add a,(ix+*)						adc a,ixh	adc a,ixl	adc a,(ix+*)	
9					sub ixh	sub ixl	sub (ix+*)						sbc a,ixh	sbc a,ixl	sbc a,(ix+*)	
A					and ixh	and ixl	and (ix+*)						xor ixh	xor ixl	xor (ix+*)	
B					or ixh	or ixl	or (ix+*)						cp ixh	cp ixl	cp (ix+*)	
C												IX BITS				
D																
E		pop ix		ex (sp),ix		push ix				jp (ix)						
F										ld sp,ix						

Figure 5: Z80 indexed (IX) instructions (source: clrhome.org)

IX bit instructions (DDCB)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	rlc (ix+*),b	rlc (ix+*),c	rlc (ix+*),d	rlc (ix+*),e	rlc (ix+*),h	rlc (ix+*),l	rlc (ix+*)	rlc (ix+*),a	rrc (ix+*),b	rrc (ix+*),c	rrc (ix+*),d	rrc (ix+*),e	rrc (ix+*),h	rrc (ix+*),l	rrc (ix+*)	rrc (ix+*),a
1	rl (ix+*),b	rl (ix+*),c	rl (ix+*),d	rl (ix+*),e	rl (ix+*),h	rl (ix+*),l	rl (ix+*)	rl (ix+*),a	rr (ix+*),b	rr (ix+*),c	rr (ix+*),d	rr (ix+*),e	rr (ix+*),h	rr (ix+*),l	rr (ix+*)	rr (ix+*),a
2	sla (ix+*),b	sla (ix+*),c	sla (ix+*),d	sla (ix+*),e	sla (ix+*),h	sla (ix+*),l	sla (ix+*)	sla (ix+*),a	sra (ix+*),b	sra (ix+*),c	sra (ix+*),d	sra (ix+*),e	sra (ix+*),h	sra (ix+*),l	sra (ix+*)	sra (ix+*),a
3	sll (ix+*),b	sll (ix+*),c	sll (ix+*),d	sll (ix+*),e	sll (ix+*),h	sll (ix+*),l	sll (ix+*)	sll (ix+*),a	srl (ix+*),b	srl (ix+*),c	srl (ix+*),d	srl (ix+*),e	srl (ix+*),h	srl (ix+*),l	srl (ix+*)	srl (ix+*),a
4	bit 0,(ix+*)	bit 0,(ix+*)	bit 0,(ix+*)	bit 0,(ix+*)	bit 0,(ix+*)	bit 0,(ix+*)	bit 0,(ix+*)	bit 0,(ix+*)	bit 1,(ix+*)	bit 1,(ix+*)	bit 1,(ix+*)	bit 1,(ix+*)	bit 1,(ix+*)	bit 1,(ix+*)	bit 1,(ix+*)	bit 1,(ix+*)
5	bit 2,(ix+*)	bit 2,(ix+*)	bit 2,(ix+*)	bit 2,(ix+*)	bit 2,(ix+*)	bit 2,(ix+*)	bit 2,(ix+*)	bit 2,(ix+*)	bit 3,(ix+*)	bit 3,(ix+*)	bit 3,(ix+*)	bit 3,(ix+*)	bit 3,(ix+*)	bit 3,(ix+*)	bit 3,(ix+*)	bit 3,(ix+*)
6	bit 4,(ix+*)	bit 4,(ix+*)	bit 4,(ix+*)	bit 4,(ix+*)	bit 4,(ix+*)	bit 4,(ix+*)	bit 4,(ix+*)	bit 4,(ix+*)	bit 5,(ix+*)	bit 5,(ix+*)	bit 5,(ix+*)	bit 5,(ix+*)	bit 5,(ix+*)	bit 5,(ix+*)	bit 5,(ix+*)	bit 5,(ix+*)
7	bit 6,(ix+*)	bit 6,(ix+*)	bit 6,(ix+*)	bit 6,(ix+*)	bit 6,(ix+*)	bit 6,(ix+*)	bit 6,(ix+*)	bit 6,(ix+*)	bit 7,(ix+*)	bit 7,(ix+*)	bit 7,(ix+*)	bit 7,(ix+*)	bit 7,(ix+*)	bit 7,(ix+*)	bit 7,(ix+*)	bit 7,(ix+*)
8	res 0,(ix+*),b	res 0,(ix+*),c	res 0,(ix+*),d	res 0,(ix+*),e	res 0,(ix+*),h	res 0,(ix+*),l	res 0,(ix+*)	res 0,(ix+*),a	res 1,(ix+*),b	res 1,(ix+*),c	res 1,(ix+*),d	res 1,(ix+*),e	res 1,(ix+*),h	res 1,(ix+*),l	res 1,(ix+*)	res 1,(ix+*),a
9	res 2,(ix+*),b	res 2,(ix+*),c	res 2,(ix+*),d	res 2,(ix+*),e	res 2,(ix+*),h	res 2,(ix+*),l	res 2,(ix+*)	res 2,(ix+*),a	res 3,(ix+*),b	res 3,(ix+*),c	res 3,(ix+*),d	res 3,(ix+*),e	res 3,(ix+*),h	res 3,(ix+*),l	res 3,(ix+*)	res 3,(ix+*),a
A	res 4,(ix+*),b	res 4,(ix+*),c	res 4,(ix+*),d	res 4,(ix+*),e	res 4,(ix+*),h	res 4,(ix+*),l	res 4,(ix+*)	res 4,(ix+*),a	res 5,(ix+*),b	res 5,(ix+*),c	res 5,(ix+*),d	res 5,(ix+*),e	res 5,(ix+*),h	res 5,(ix+*),l	res 5,(ix+*)	res 5,(ix+*),a
B	res 6,(ix+*),b	res 6,(ix+*),c	res 6,(ix+*),d	res 6,(ix+*),e	res 6,(ix+*),h	res 6,(ix+*),l	res 6,(ix+*)	res 6,(ix+*),a	res 7,(ix+*),b	res 7,(ix+*),c	res 7,(ix+*),d	res 7,(ix+*),e	res 7,(ix+*),h	res 7,(ix+*),l	res 7,(ix+*)	res 7,(ix+*),a
C	set 0,(ix+*),b	set 0,(ix+*),c	set 0,(ix+*),d	set 0,(ix+*),e	set 0,(ix+*),h	set 0,(ix+*),l	set 0,(ix+*)	set 0,(ix+*),a	set 1,(ix+*),b	set 1,(ix+*),c	set 1,(ix+*),d	set 1,(ix+*),e	set 1,(ix+*),h	set 1,(ix+*),l	set 1,(ix+*)	set 1,(ix+*),a
D	set 2,(ix+*),b	set 2,(ix+*),c	set 2,(ix+*),d	set 2,(ix+*),e	set 2,(ix+*),h	set 2,(ix+*),l	set 2,(ix+*)	set 2,(ix+*),a	set 3,(ix+*),b	set 3,(ix+*),c	set 3,(ix+*),d	set 3,(ix+*),e	set 3,(ix+*),h	set 3,(ix+*),l	set 3,(ix+*)	set 3,(ix+*),a
E	set 4,(ix+*),b	set 4,(ix+*),c	set 4,(ix+*),d	set 4,(ix+*),e	set 4,(ix+*),h	set 4,(ix+*),l	set 4,(ix+*)	set 4,(ix+*),a	set 5,(ix+*),b	set 5,(ix+*),c	set 5,(ix+*),d	set 5,(ix+*),e	set 5,(ix+*),h	set 5,(ix+*),l	set 5,(ix+*)	set 5,(ix+*),a
F	set 6,(ix+*),b	set 6,(ix+*),c	set 6,(ix+*),d	set 6,(ix+*),e	set 6,(ix+*),h	set 6,(ix+*),l	set 6,(ix+*)	set 6,(ix+*),a	set 7,(ix+*),b	set 7,(ix+*),c	set 7,(ix+*),d	set 7,(ix+*),e	set 7,(ix+*),h	set 7,(ix+*),l	set 7,(ix+*)	set 7,(ix+*),a

Figure 6: Z80 indexed (IX) bit manipulation operations (source: clrhome.org)

Timings

If you are about to write a ZX Spectrum emulator—or any computer emulator—you soon learn that taking care of timing is probably the most

important thing. Without this, you won't be able to create a high-fidelity emulation of real hardware.

Note: In the future articles in this series I will treat particular aspects of timings in almost every post.

The Z80 CPU executes instructions as a series of subsequent *machine cycles*. To understand how it works, **Figure 7** gives you the detailed timing of the **`INC (HL)`** instruction. **`INC (HL)`** increments the value stored at the memory address pointed by the **HL** register pair.

As the figure shows, the CPU executes this instruction in four machine cycles:

- **M1**: The CPU reads the opcode from the memory address pointed by **PC** (Program Counter). The execution logic understands what this instruction means, and how to process it.
- **M2**: The CPU reads the contents of the memory address pointed by **HL** into some internal ALU register to be ready to process it.
- **M3**: The CPU increments the value of the internal ALU register
- **M4**: The CPU writes back the incremented value to the memory address pointed by **HL**.

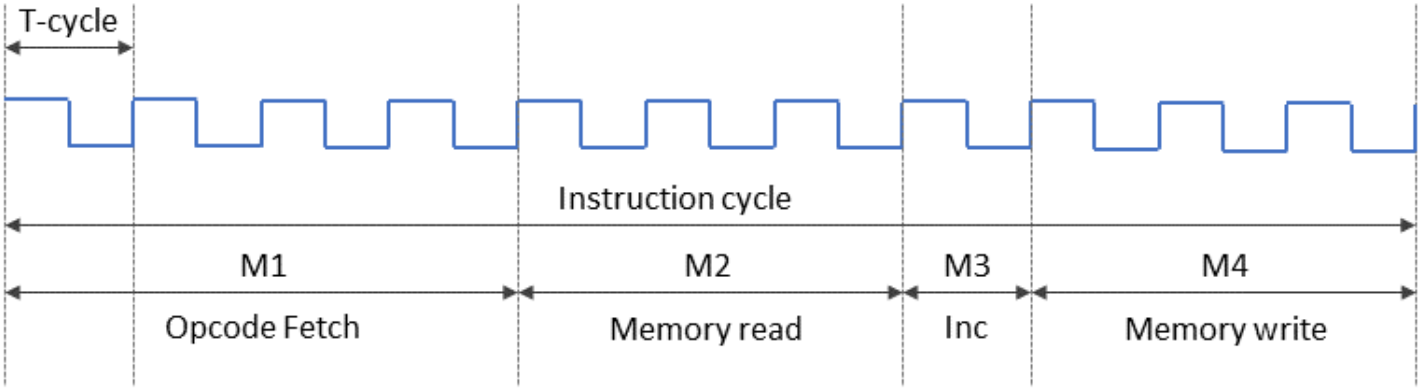

*Figure 7: Timing diagram of the **`INC (HL)`** instruction*

Well, this concise description of the **`M1`**...**`M4`** cycles did not mention many subtle operation details. The real execution is more complicated. The diagram shows that the machine cycles utilize more than a single clock pulse (T-cycle) to carry out their tasks. The longest is **`M1`** with four T-cycles. Reading and writing memory takes three T-cycles, respectively. The fastest step is the increment, it consumes a single clock cycle.

The **`M1`** cycle is a special one, so I'd like to add some more details on it:

Every instruction starts with fetching the operation code. If the code is prefixed (such as in case of extended, indexed, and bit manipulation instructions), every opcode fetch is similar. The **`M1`** cycle takes for T-cycles, **`T1`**, **`T2`**, **`T3`**, and **`T4`**. This is what happens during **`M1`**:

- As **`T1`** begins, the CPU puts the contents of PC to the address bus. About a half T-cycle later, the CPU sign the **`MREQ`** (*Memory Request*) control signal in tandem with **`RD`** (*Read*).
- In **`T2`**, the memory responds by placing the content of the memory addressed by PC to the data bus. (By this time, the memory address

stabilizes on the address bus.)

- Just as **T3** begins, with the rising edge of the clock signal, the CPU reads the contents of the data bus—the opcode gets into one of the internal registers. The CPU revokes the **RD**, and **MREQ** signals and puts lower seven bit of **R** (Refresh Page Register) to the lower seven lines of the address bus. At the same time, the CPU places the contents of **I** (Interrupt Register), to the highest eight lines of the address bus, and signs **RFSH** (Refresh signal).
- During **T3** and **T4**, when the refresh page address is stabilized on the address bus, the CPU raises the **MREQ** signal again. The combination of **MREQ** and **RFSH** allows the DRAM chips to refresh the memory contents of the addressed page.
- By the end of **T3**, the CPU analyzes the opcode read and prepares for the subsequent machine cycles. Many operations, are simple and do not need any further memory or I/O access. The CPU executes these operations during **T4**.
- By the end of **T4**, **MREQ**, **RD** (and **RFSH**, a little bit later) go back to their inactive state. The CPU increments the last seven bits of **R** while keeping its most significant bit.

As you can see, the **M1** machine cycle is significant, it is responsible for refreshing the DRAM memory. Without this periodic refresh cycle, the memory would forget its content. Just to have an idea about this time, every page should be refreshed in every 64 milliseconds or less.

To let peripherals and other devices know that the CPU executes **M1**, Z80 has a system control signal, **M1**, that goes active during **T1** and **T2**.

It may happen that during memory and I/O operations the CPU needs to wait while the memory or a device gets ready for a data transfer. The CPU has a **WAIT** input signal; devices may use it to sign that they are not prepared to let the CPU carry on the read or write operation.

> **Note**: Later, in another post, you will learn that the **$4000**–**$7FFF** range of memory in ZX Spectrum 48K is contended. Sometimes, when the CPU wants to read or write the memory, it's forced to **WAIT**, as the ULA has priority to keep the electron ray in the cathode tub uninterrupted.

Memory read and write, I/O read and write machine cycles have their detailed timings—similarly to **M1**. Here I won't detail them. If you are interested, check the official Zilog Z80 documentation here.

Interrupts

The CPU cannot continuously poll devices whether they have something to tell. Devices can notify the CPU by generating an interrupt signal. Z80 receives that signal and suspends its normal execution. In response, it executes a routine, the interrupt routine. As that routine is completed, the CPU goes back and continues executing instructions right from the point it was before receiving the signal.

Z80 can handle two kinds of interrupts through the **`INT`** and **`NMI`** signals. **`NMI`** stands for Non-Maskable Interrupt. When the CPU receives an **`NMI`** request, it executes the interrupt routine starting at address **`$0066`**.

`INT` raises a maskable interrupt. Maskable means that you can disable (and re-enable) it from software—with the **`DI`** (*disable interrupt*), and **`EI`** (*enable interrupt*) Z80 instructions, respectively.

Z80 offers three interrupt handling modes. You can activate these with the **`IM 0`**, **`IM 1`**, and **`IM 2`** instructions:

- **`IM 0`**: In this mode the interrupting device can force the CPU to execute a single machine operation. The device places the opcode to the address bus to let the CPU read it in. If the operation contains more bytes, the device should take care to provide those bytes according to the normal memory read timing sequence. *No ZX Spectrum models use this interrupt mode.*
- **`IM 1`**: This is the simplest interrupt mode. When the CPU receives the request, it starts the interrupt routine at address **`$0038`**. *By default, all ZX Spectrum models—I mean, their operating system—uses this mode.*
- **`IM 2`**: This mode is the most complex, often called *vectored interrupt*, for it allows an indirect call to any memory location by an 8-bit vector supplied from the peripheral device. This vector then becomes the least significant eight bits of the indirect pointer, while **`I`** (Interrupt Register) in the CPU provides the most significant eight bits. This address points to an address in a vector table that is the starting

address for the interrupt service routine. *ZX Spectrum games often use this mode to change the original interrupt handler routine entirely.*

Of course, the CPU cannot stash its current operation at the very moment an interrupt request arrives. Z80 defines an interrupt request/acknowledge timing cycle:

The CPU samples the **INT** signal with the rising edge of the final clock at the end of any instruction. The signal is not accepted unless the maskable interrupt is enabled. When the signal is accepted, the CPU generates a special **M1** cycle. During this cycle, the **IORQ** signal becomes active (instead of the normal **MREQ**) to indicate that the interrupting device can place an 8-bit vector on the data bus. Two wait states are automatically added to this cycle. These states are added so that a *ripple priority interrupt scheme* can be easily implemented. The two wait states allow sufficient time for the ripple signals to stabilize and identify which I/O device must insert the response vector.

Next: Emulating the Z80A CPU

By now, you know enough information about the Z80 CPU. You are prepared to understand the operation of the Z80 emulator. In the next posts of this series, you will learn about the concepts, design, implementation, and testing details.

← Prev Post Next Post →

1 Comment

GOING HERE AUGUST 10, 2019

REPLY

It’s hard to find educated people in this particular topic, but you sound like you know what you’re talking about! Thanks

Leave a comment

Comment…

* Name…

* Email…

Website…

SUBMIT NOW