

Operating Systems

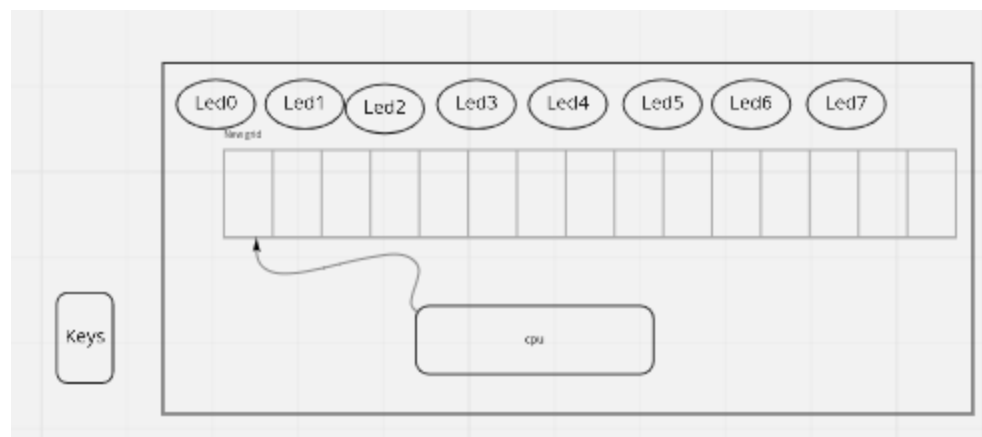
Ahmad Yoosofan

University of Kashan

<https://yoosofan.github.io/en/>

Yoosofan Imaginary Computer

Based on Morris Mano's famous book



Instruction Set(I)

AND: Logical AND memory with AC
ADD: Arithmetic ADD memory with AC
LDA: Load from memory to AC
STA: Store AC to memory
BUN: Branch unconditional
ISZ: Increment and skip if zero
CLA: Clear AC
CLE: Clear E
CMA: Complement AC
CME: Complement E
CIR: Circulate right (AC and E)
CIL: Circulate left (AC and E)

INC: Increment AC
SPA: Skip if positive AC
SNA: Skip if negative AC
SZA: Skip if zero AC
SZE: Skip if zero E
HLT: Halt
OUT: Output a character from AC
SKO: Skip if output flag
NOP: No operation

Instruction Set Binary(I)

AND:	00001
ADD:	00010
LDA:	00011
STA:	00100
BUN:	00101
ISZ:	00110
CLA:	00111
CLE:	01000
CMA:	01001
CME:	01010
CIR:	01011
CIL:	01100

INC:	01101
SPA:	01110
SNA:	01111
SZA:	10000
SZE:	10001
HLT:	10010
OUT:	10011
SKO:	10100
NOP:	10101

hex pad connect to microcontroller

<https://www.circuitstoday.com/interfacing-hex-keypad-to-8051>

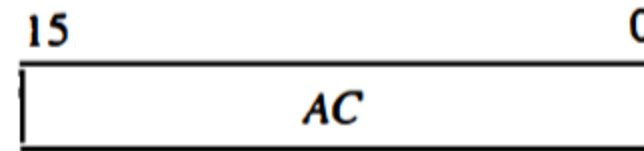
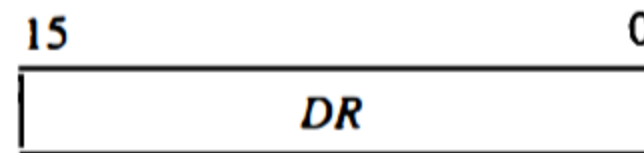
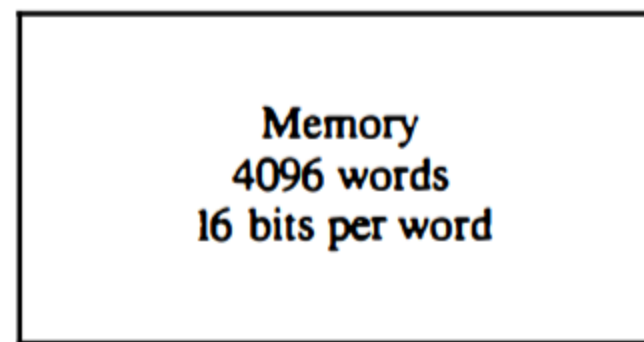
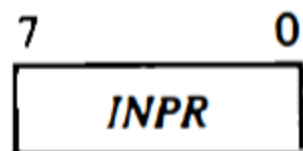
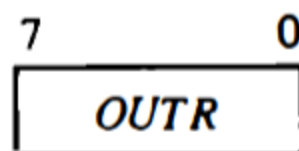
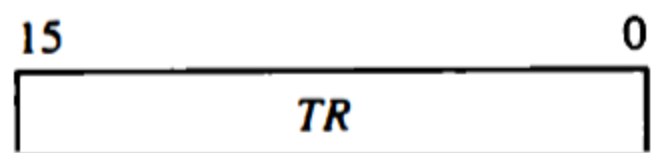
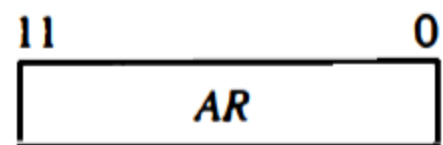
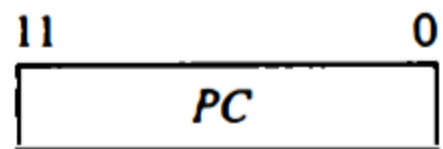
<https://circuitdigest.com/microcontroller-projects/keypad-interfacing-with-avr-atmega32>

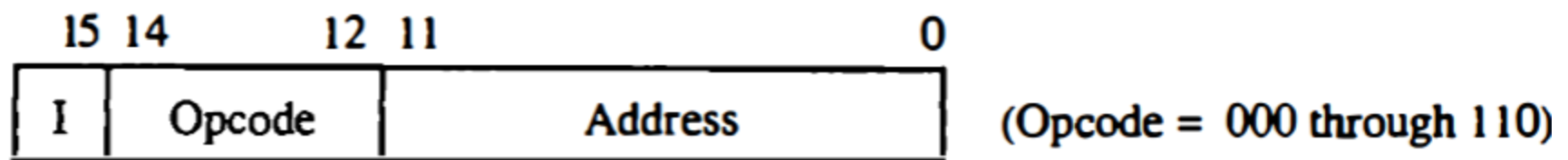
Example

- <https://github.com/yoosofan/mano-computer-simulator-js>
- <https://yoosofan.github.io/mano-computer-simulator-js/>
- <https://github.com/Naheel-Azawy/Simple-Computer-Simulator/blob/master/test/test-symbolic>
- <https://github.com/Naheel-Azawy/Simple-Computer-Simulator/blob/master/test/test>
- <https://github.com/Naheel-Azawy/Simple-Computer-Simulator/tree/master/test>

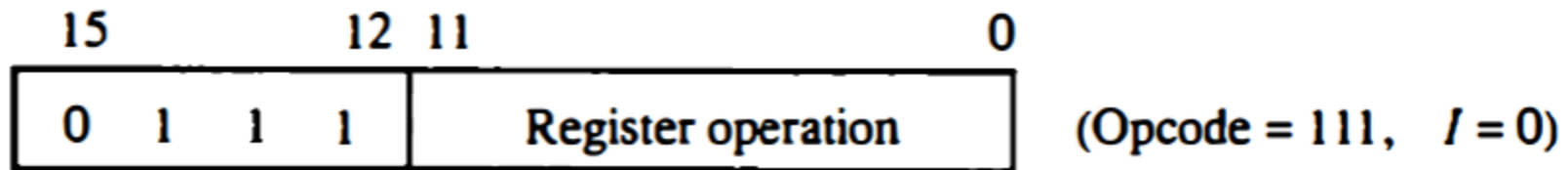
Other assembly

- <http://imrannazar.com/arm-opcode-map>
- <https://iitd-plos.github.io/col718/ref/arm-instructionset.pdf>
- https://wiki.osdev.org/X86-64_Instruction_Encoding
- https://cs.brown.edu/courses/cs033/docs/guides/x64_cheatsheet.pdf
- <https://sites.google.com/site/nttrungmtwiki/home/rce/assembly-language/x64-opcode-and-instruction-reference-home>
- <http://ref.x86asm.net/coder64.html>
- arm 32 opcodes
- <http://z80-heaven.wikidot.com/instructions-set:ld>
- <http://z80-heaven.wikidot.com/opcode-reference-chart>
- <https://smallcomputercentral.files.wordpress.com/2017/12/asm80-com-tutorial-e1-0-01.pdf>
- <https://stackoverflow.com/questions/22838444/convert-an-8bit-number-to-hex-in-z80-assembler>
- <https://www.vcfed.org/forum/forum/technical-support/vintage-computer-programming/76419-z80-hello-world-example-in-hex>
- <https://www.cemetech.net/forum/viewtopic.php?t=15710&start=0>
- z80 assembly codes

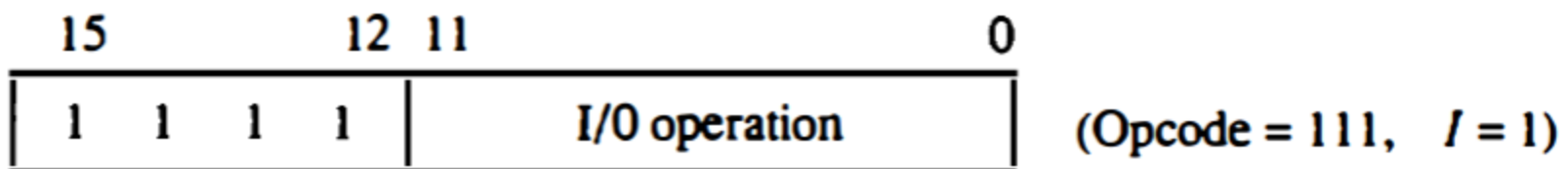




(a) Memory – reference instruction



(b) Register – reference instruction



(c) Input – output instruction

00101	00000	1010
00110	00000	1100
00111	00000	1110
01000		
00000		

اگر حداکثر ۳۲ دستور داشته باشیم پس پنج بیت برای دستورهای نیاز داریم برای سادگی فرض می‌کنیم که طول همهٔ دستورها یکسان است یعنی هم دو بایت را می‌گیرند فرض کنید دستورهای پنج بیت نیاز دارند پس ۱۱ بیت برای آدرس

حداکثر حافظهٔ این کامپیوتر چقدر می‌تواند باشد. اگر بخواهیم بایستی آدرس دهی کنیم

$$2^{11} = 2\text{kB}$$

B = Byte

اگر آدرس‌دهی را دو بایتی در نظر بگیریم

$$4\text{kB (word = 2 byte)}$$

Output

LED

seven segment

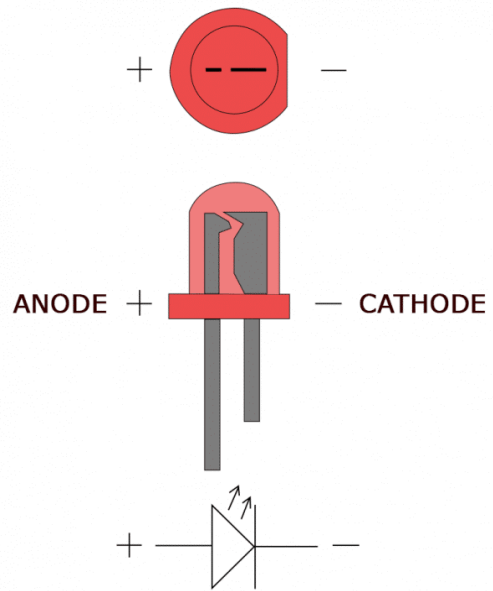
- <https://www.rapidtables.com/convert/number/hex-to-binary.html>
- convert hex to binary
- <https://clrhome.org/asm/>

Output Problem

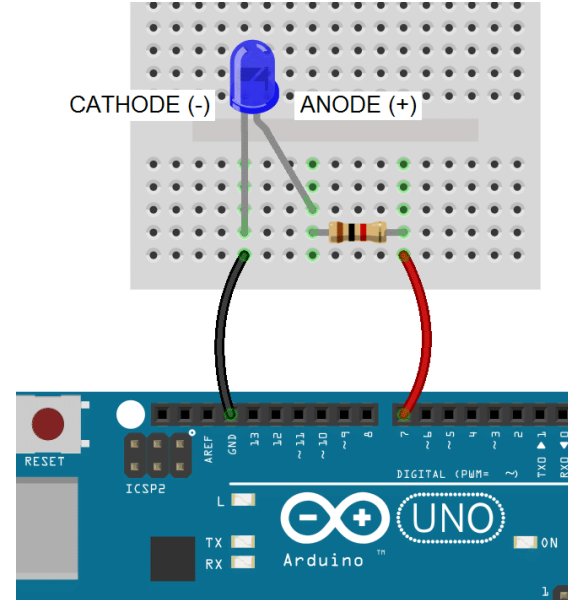
```
lda a
add b
sta c
out
hlt
a, 5
b, 2
c, 0
```

```
.....
.....
LB1: out
      sko
      bun LB1
.....
.....
```

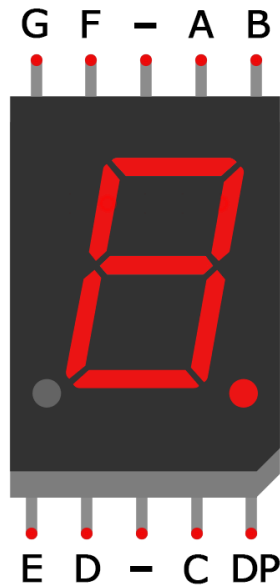
Simple LED



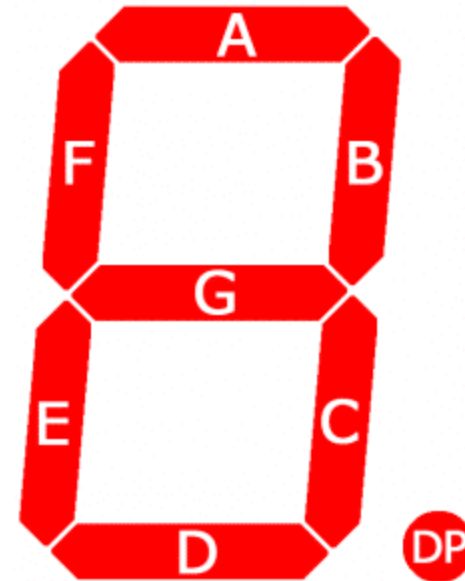
[circuitbasics](http://circuitbasics.com)



Seven segment display



[circuitbasics](#) [askingthelot](#)



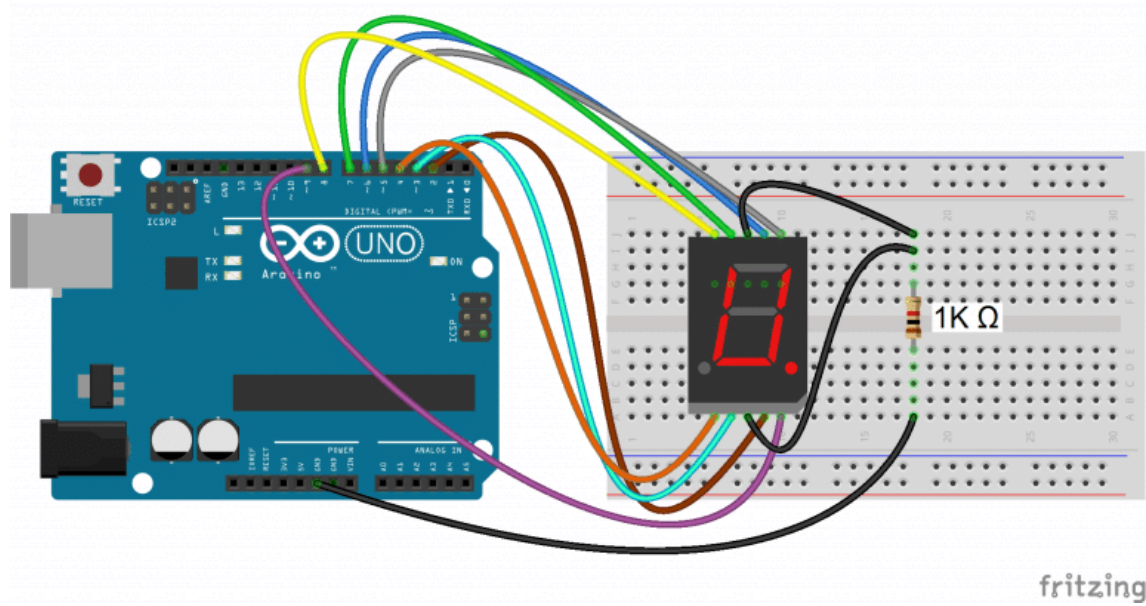
[youtu.be](#) [element14](#)

Imaginary Computer

- Consider it as real a computer
- Think about business plan
- Consider customers' need
- Consider other companies

YIC 30

Seven segment

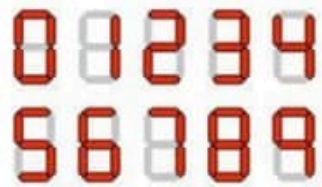
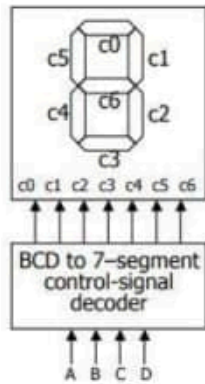


[circuitbasics](http://circuitbasics.com)

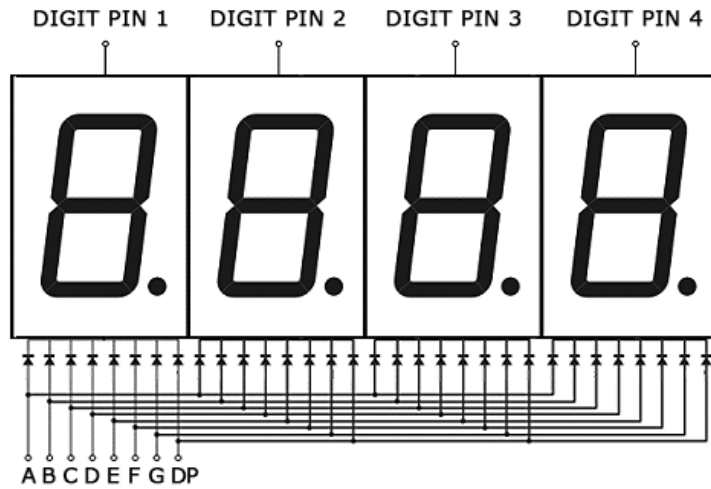
Issues of YIC 30

- Convert binary number to 7 segment code
- Old codes only LED
- LED & seven segment
- Changing CPU
- Cost of changes
- Just one 7 segment ?
- for every output, seven segment code should be added

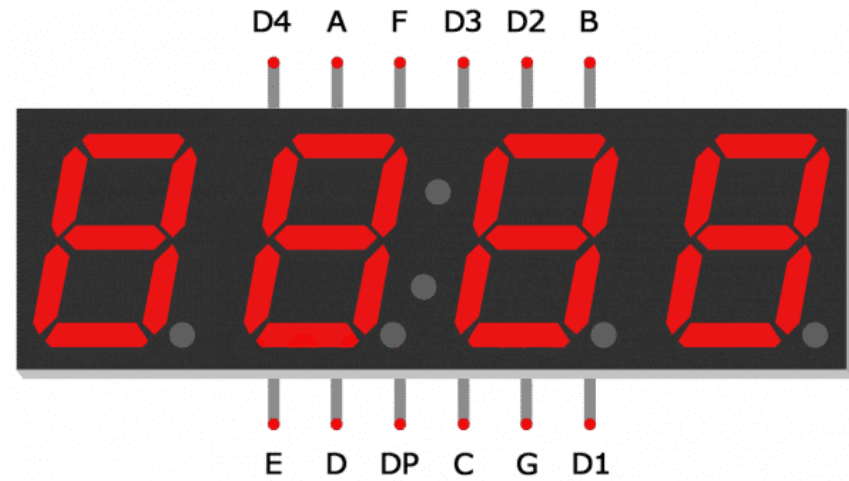
Hardware instead of Software

[illegible]

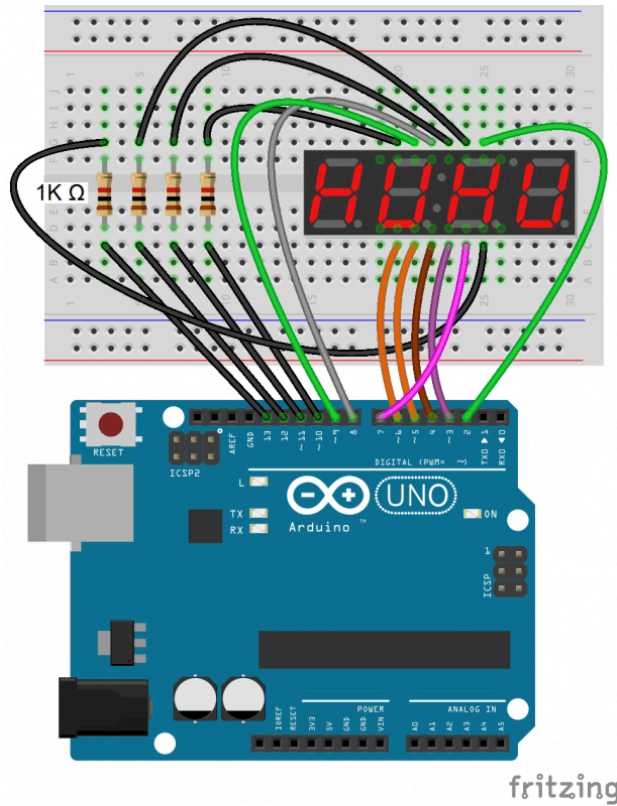
4 Digit 7-Segment Displays



[circuitbasics](http://circuitbasics.com)



Connecting 4 Digit 7-Segment Displays



Arduino Print 4 to 7-segment

```
#include "SevSeg.h"
SevSeg sevseg;

void setup(){
  byte numDigits = 1;
  byte digitPins[] = {};
  byte segmentPins[] =
    {6, 5, 2, 3, 4, 7, 8, 9};
  bool resistorsOnSegments = true;

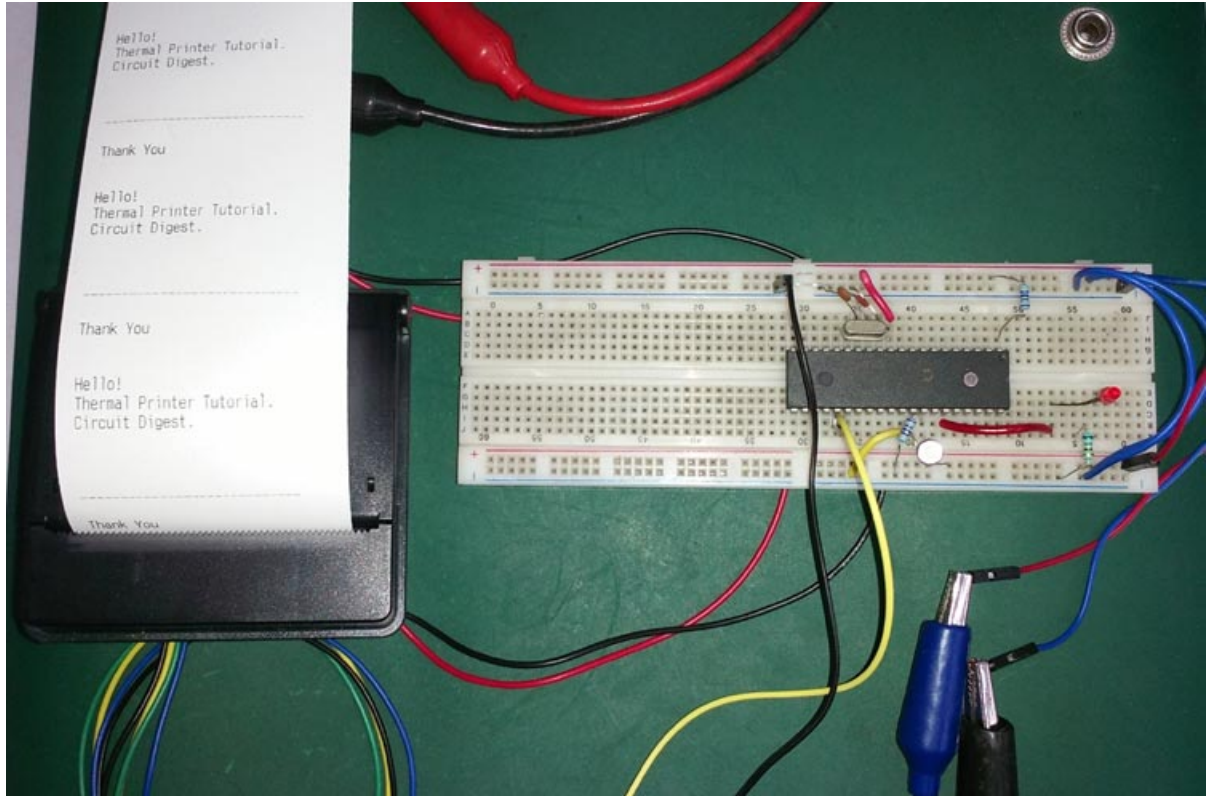
  byte hardwareConfig = COMMON_CATHODE;
  sevseg.begin(hardwareConfig,
    numDigits, digitPins, segmentPins,
    resistorsOnSegments
  );
  sevseg.setBrightness(90);
}

void loop(){
  sevseg.setNumber(4);
  sevseg.refreshDisplay();
}
```

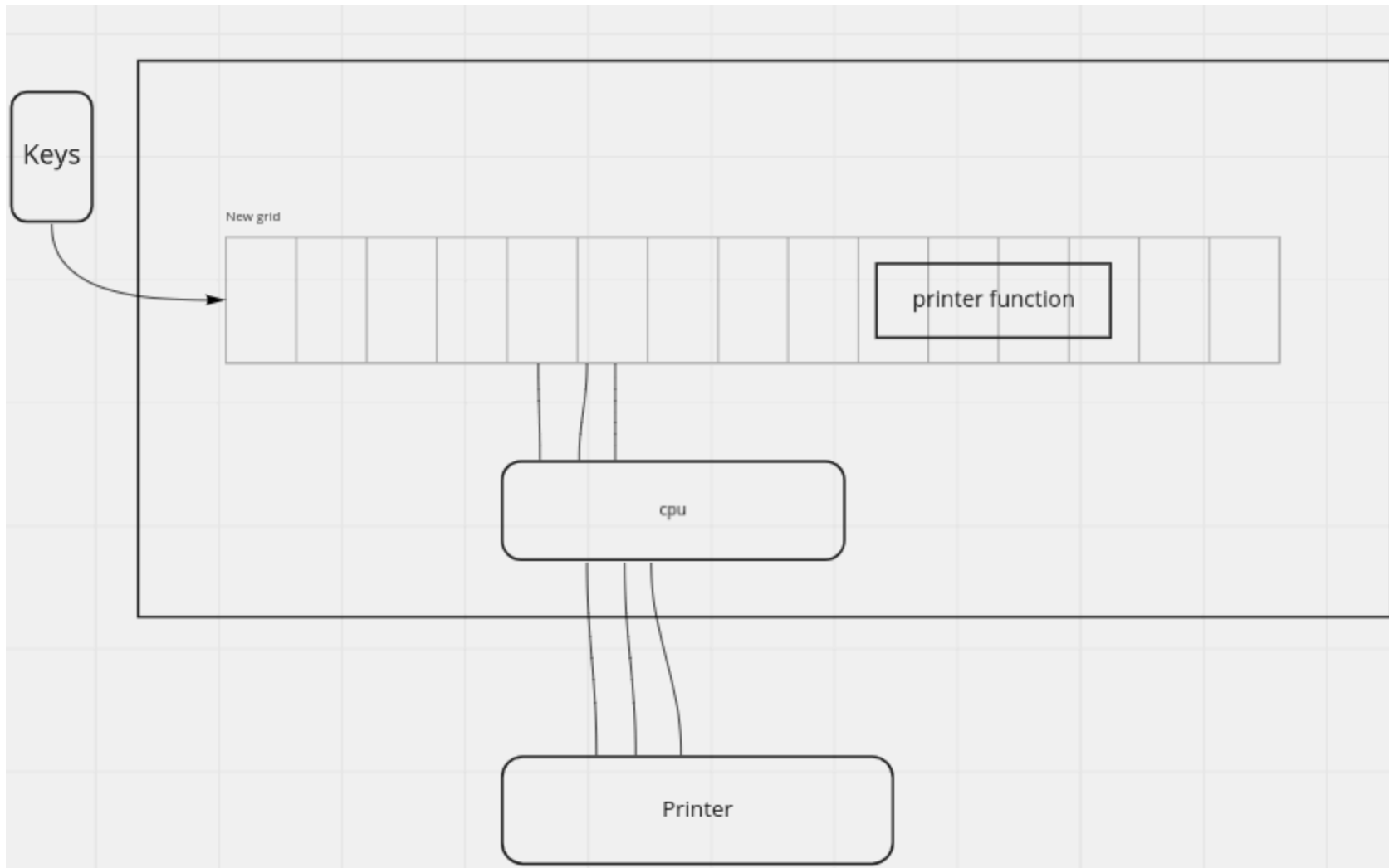
Segment Pin	Arduino Pin
A	6
B	5
C	2
D	3
E	4
F	7
G	8
DP	9

YIC 40 - BSA

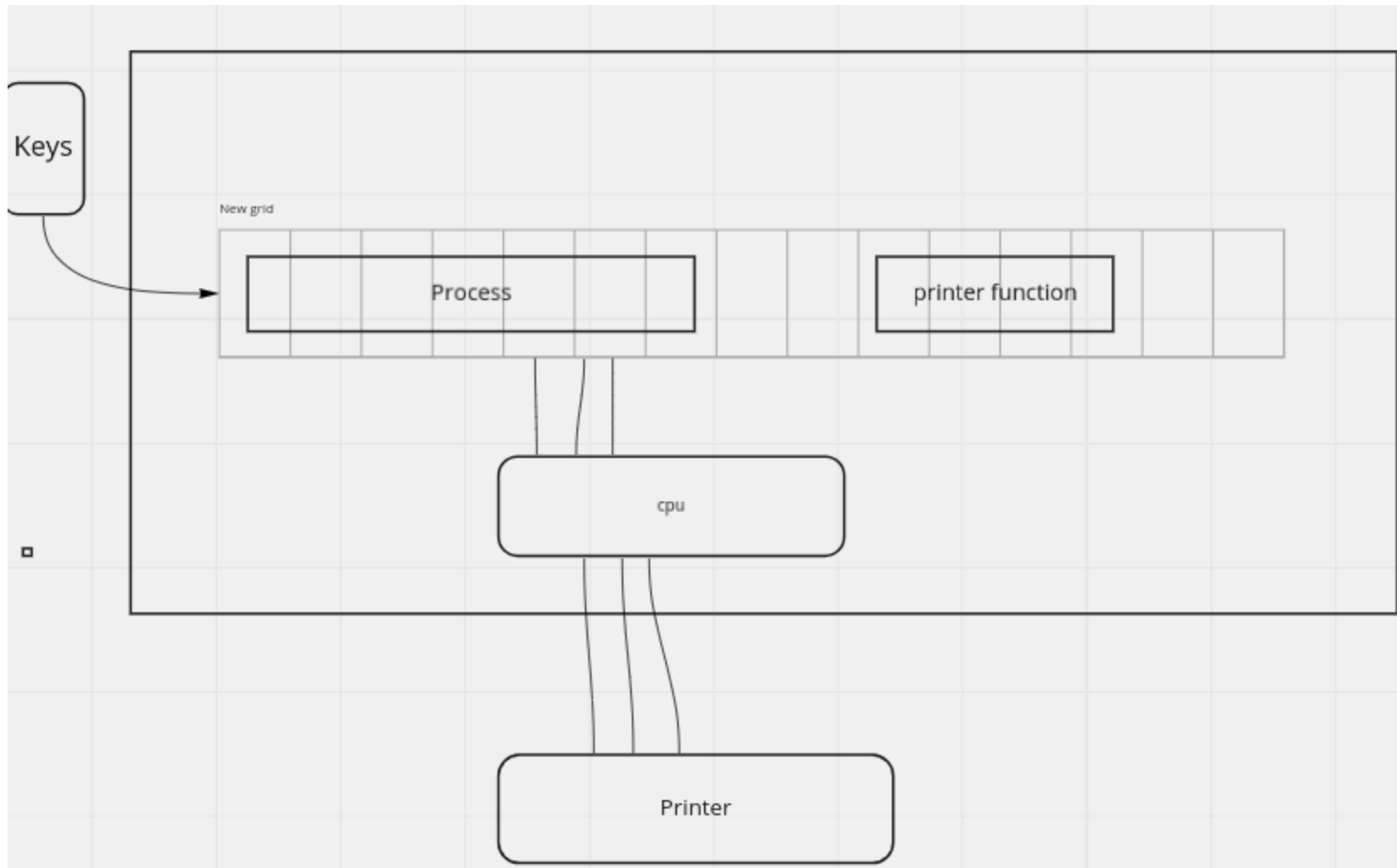
1. LED output code
2. 7 segment code
3. Printer
4. output selector
5. Adding porecedures
6. Device Drivers
7. Adding more devices
8. No error checking
9. Send data
10. API (protocol)



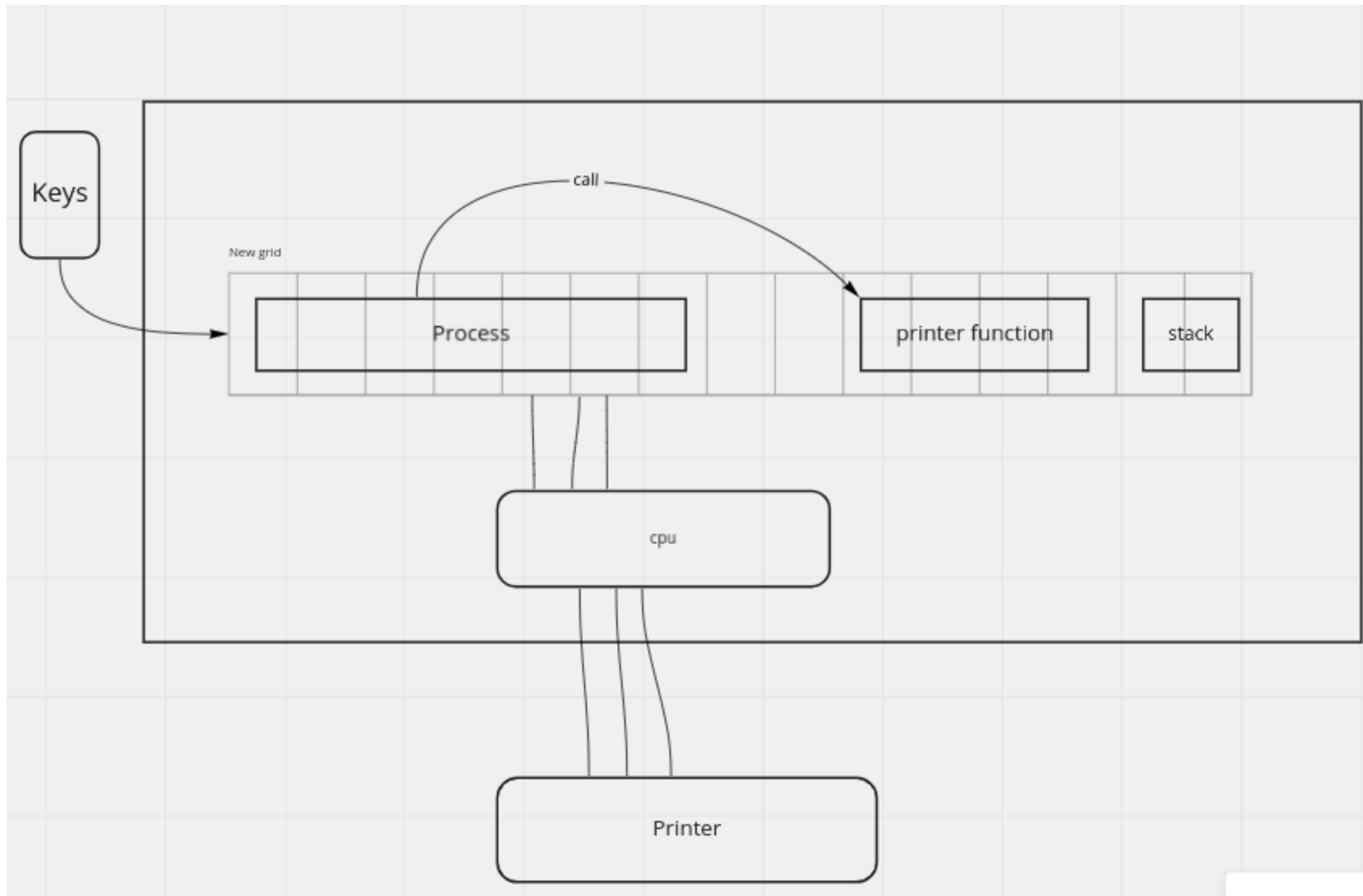
Printer Function



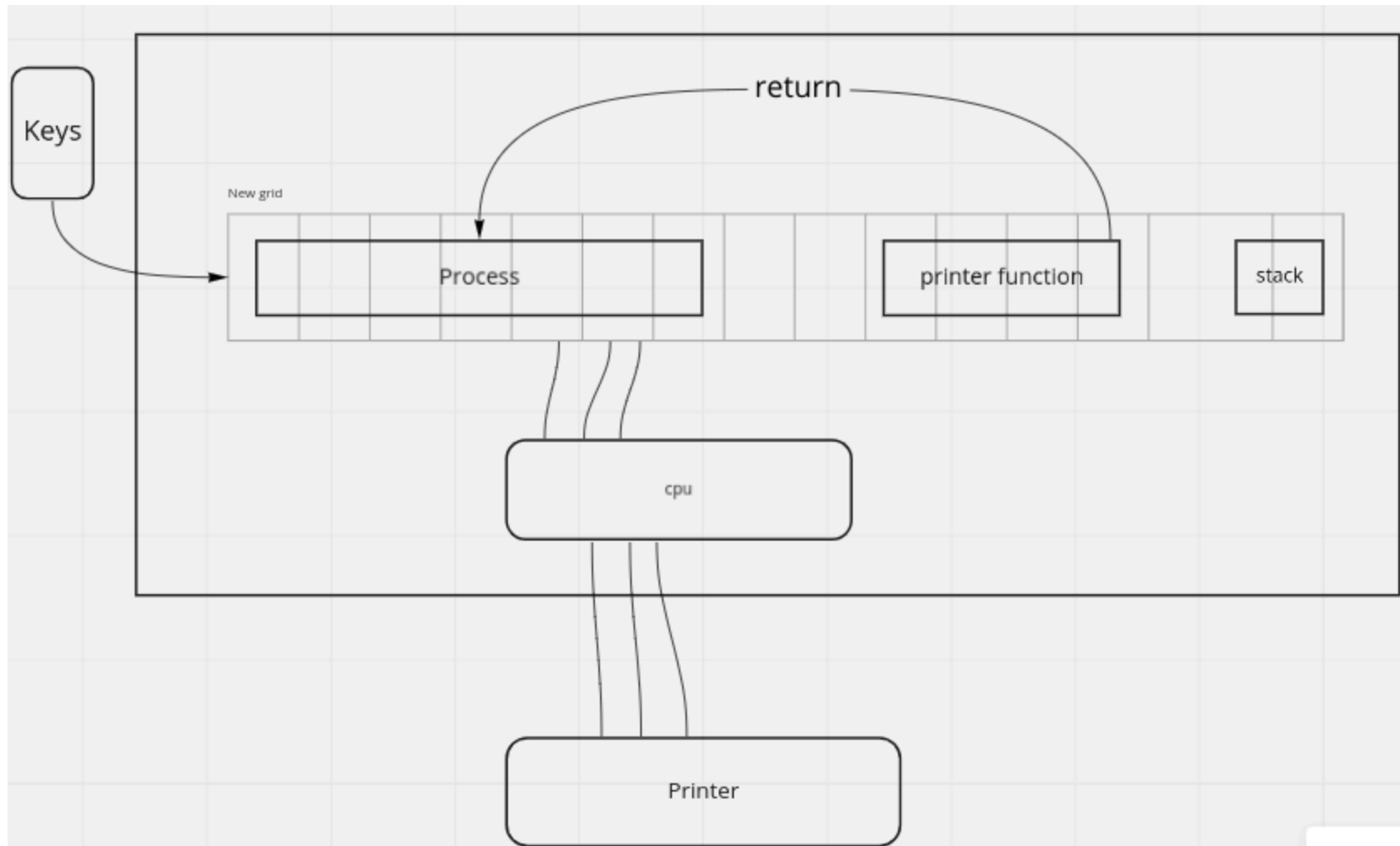
Process along Printer Function



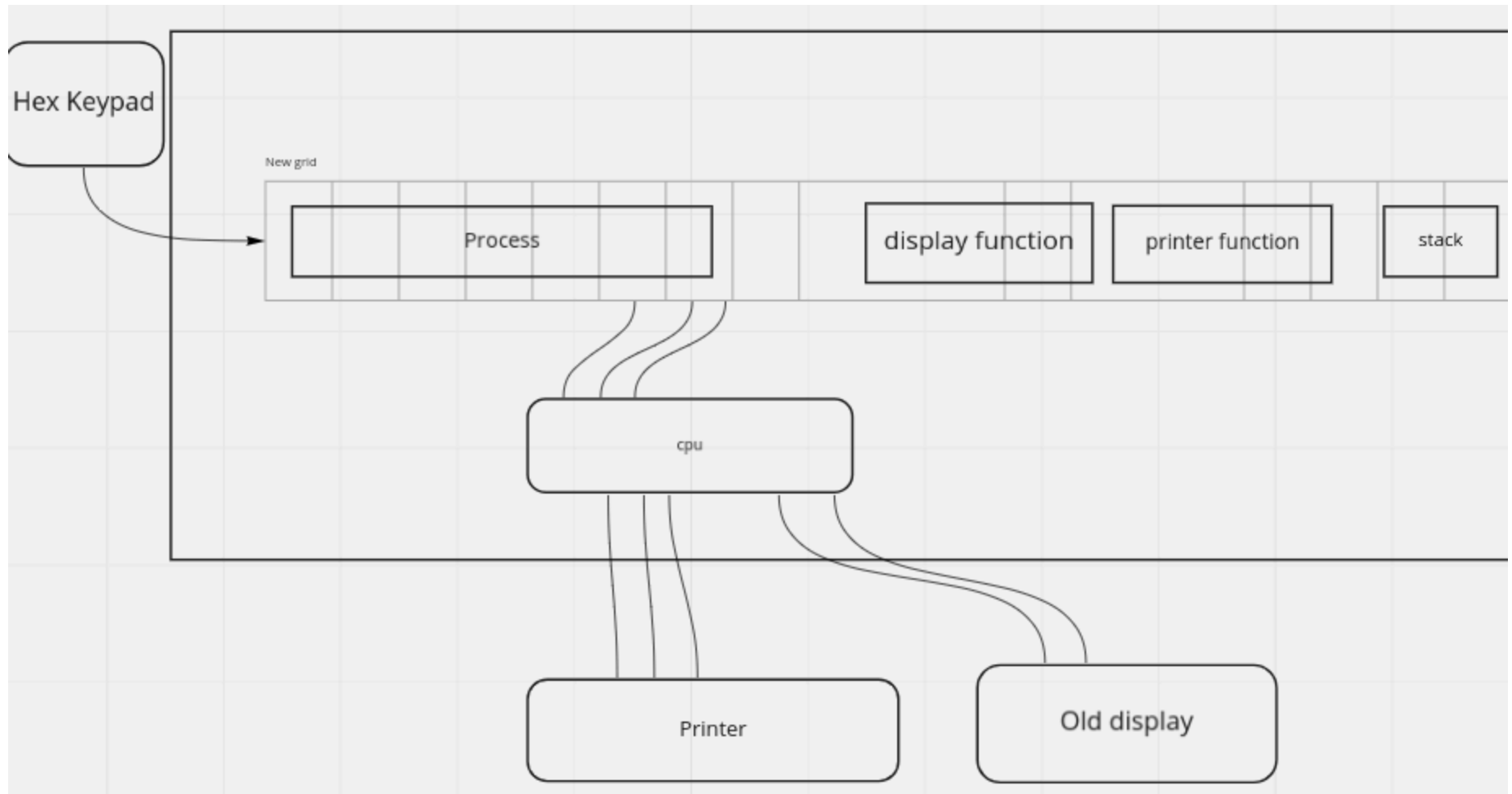
Jump to Printer Procedure



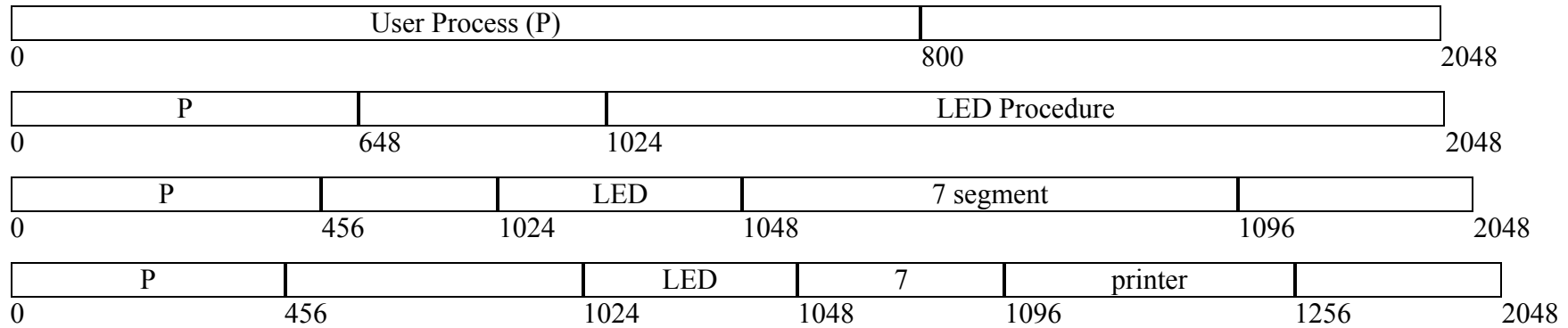
Return from Printer Procedure



Display and Printer Procedure

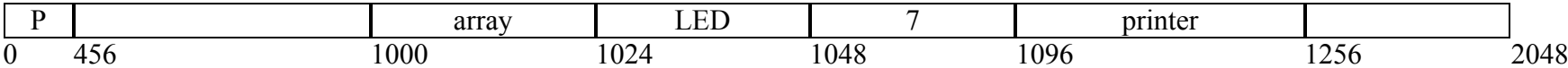


Adding procedurs to memory

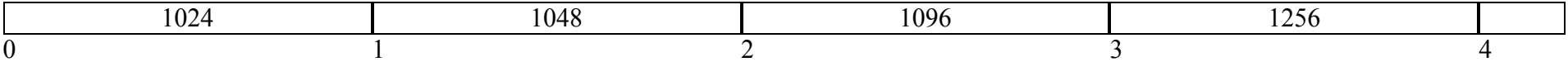


Users (programmers) should know where these precedures are

YIC50 - Array of Addresses



Array



YIC60 - Input Devices

- Card Reader
- Necessary Loops
- Check Errors
- Polling Method
- Hollerith and IBM keypunches, 1890
- IBM 011 Electric Key Punch(1923)
- IBM Type 032 Printing Punch(1935)
- A Key Punch Room in the 1960s





```
ORG 0
START, BSA READ1
      STA BYTE1
      LDA BYTE1
      BSZ OUTCH
      HLT

READ1, HEX 0
RDCNT, SKI
      BUN RDCNT
      INP
      BUN (READ1)

OUTCH, HEX 0
      OUT
      BUN (OUTCH)

BYTE1, DEC 0
      END
```

Simple Input Output Code - YIC60

```
1      ORG      0
2  START,  BSA   INPUT
3          STA   NUM1
4
5          BSA   INPUT
6          STA   NUM2
7
8          LDA   NUM1
9          ADD   NUM2
10         STA   RESULT
11
12         BSA   OUTPUT
13
14         HLT
```

```
15  INPUT,  HEX      0
16  POLL,   SKI
17          BUN      POLL
18          INP
19          BUN      (INPUT)
20
21  OUTPUT,  HEX      0
22  OUT_P,   SK0
23          BUN      OUT_P
24          OUT
25          BUN      (OUTPUT)
26
27  // Data Section
28  NUM1,    DEC      0
29  NUM2,    DEC      0
30  RESULT,  DEC      0
31
32          END
```


YIC60 - Array of Addresses

P		array	LED	7	printer		
0	456	1000	1024	1048	1096	1256	2048

Array

1024	1048	1096	1256	
0	1	2	3	4

YIC60 code with array(I)

```
1  START      BSA      (PVT_INPUT)
2              STA      NUM1
3              BSA      (PVT_INPUT)
4              STA      NUM2
5              LDA      NUM1
6              ADD      NUM2
7              STA      RESULT
8              BSA      (PVT_OUTPUT)
9              HLT
10 NUM1,      DEC      0
11 NUM2,      DEC      0
12 RESULT,    DEC      0
13
14              ORG      150
15 PVT_INPUT,  HEX      200
16 PVT_OUTPUT, HEX      220
17
18              ORG      200
19 INPUT,      HEX      0
20 IN_POLL,    SKI
21              BUN      IN_POLL
22              INP
23              BUN      (INPUT)
```

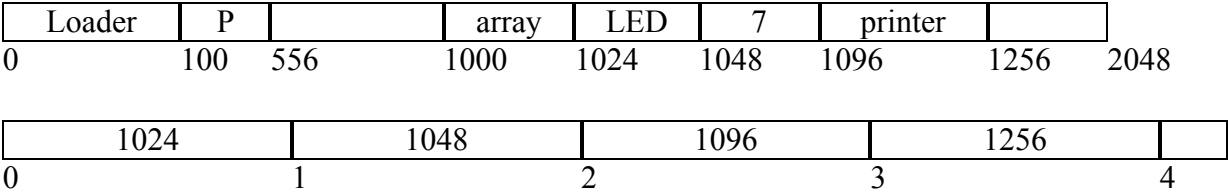
```
19              ORG      220
20 OUTPUT,    HEX      0
21              STA      TEMP_OUT
22
23 OUT_POLL,   SKO
24              BUN      OUT_POLL
25              LDA      TEMP_OUT
26              OUT
27              BSA      (PVT_DELAY)
28              BUN      (OUTPUT)
29
30              ORG      240
31 DELAY,      HEX      0
32              LDA      DELAY_COUNT
33              STA      DELAY_CTR
34
35 DELAY_LOOP,
36              LDA      DELAY_CTR
```

YIC60 code with array(II)

```
1  DELAY,  HEX      0
2          LDA      DELAY_COUNT
3          STA      DELAY_CTR
4
5  DELAY_LOOP,
6          LDA      DELAY_CTR
7          SZA
8          BUN      CONTINUE_DELAY
9          BUN      (DELAY)
10
11 CONTINUE_DELAY,
12         DEC
13         STA      DELAY_CTR
14         BUN      DELAY_LOOP
15
16         ORG      300
17 TEMP_OUT,  DEC      0
18 DELAY_COUNT, DEC     10
```

```
21 DELAY_CTR,  DEC      0
22 PVT_DELAY,  HEX     52
23          END
```

YIC70 - Adding Loader



Array

1	START,	BSA	READ_COUNT
2		STA	BYTE_COUNT
3		BSA	INIT_LOAD
4		BSA	LOAD_LOOP
5		BSA	EXECUTE
6		BUN	0
7		HLT	
8			
9	READ_COUNT,	HEX	0
10	RD_CNT,	SKI	
11		BUN	RD_CNT
12		INP	
13		BUN	READ_COUNT I
14			
15	INIT_LOAD,	HEX	0
16		LDA	ZERO
17		STA	LOAD_PTR
18		STA	CURRENT_IDX
19		BUN	INIT_LOAD I
20			
21	LOAD_LOOP,	HEX	0
22		LDA	CURRENT_IDX

21		SUB	BYTE_COUNT
22		SPA	
23		BUN	LOAD_BYTE
24		BUN	LOAD_LOOP I
25			
26	LOAD_BYTE,		
27		BSA	READ_BYTE
28		STA	TEMP_BYTE
29			
30		LDA	LOAD_PTR
31		STA	STORE_PTR
32		LDA	TEMP_BYTE
33		STA	STORE_PTR I
34			
35		LDA	LOAD_PTR
36		INC	
37		STA	LOAD_PTR

```

1      STA      LOAD_PTR
2      LDA      CURRENT_IDX
3      INC
4      STA      CURRENT_IDX
5      BUN      LOAD_LOOP
6
7  READ_BYTE,  HEX 0
8  RD_BYT,  SKI
9          BUN      RD_BYT
10         INP
11         BUN      READ_BYTE I
12
13         ORG      128
14 EXECUTE,  HEX 0
15
16         LDA      ZERO
17         STA      RESULT
18         BUN      EXECUTE I
19
20         ORG      64
21 ZERO,    DEC      0

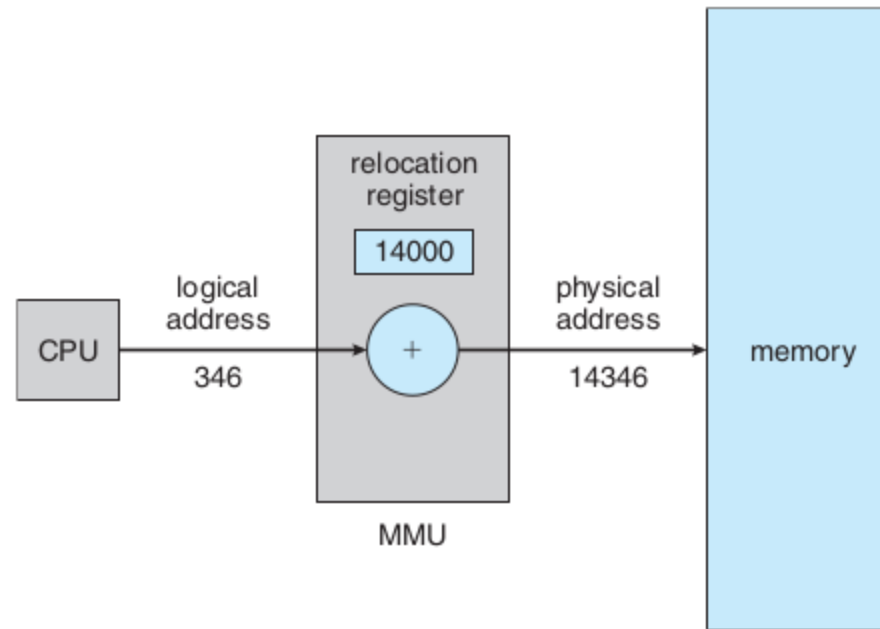
```

```

21 BYTE_COUNT, DEC      0
22 LOAD_PTR,   HEX      128
23 CURRENT_IDX, DEC      0
24 TEMP_BYTE,  DEC      0
25 STORE_PTR,  HEX      0
26 RESULT,     DEC      0
27
28         END

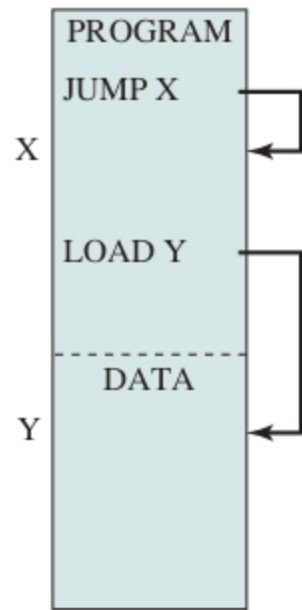
```

YIC75 Relative Address



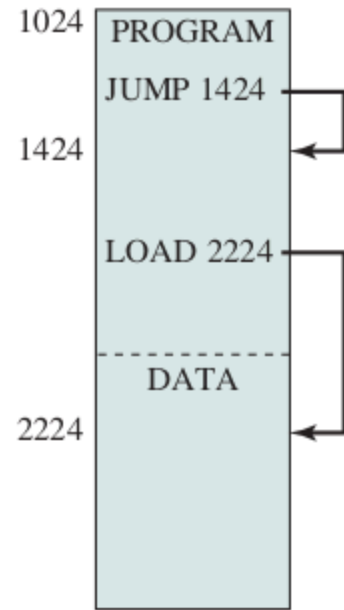
address binding, absolute and relocate loader

Symbolic
addresses



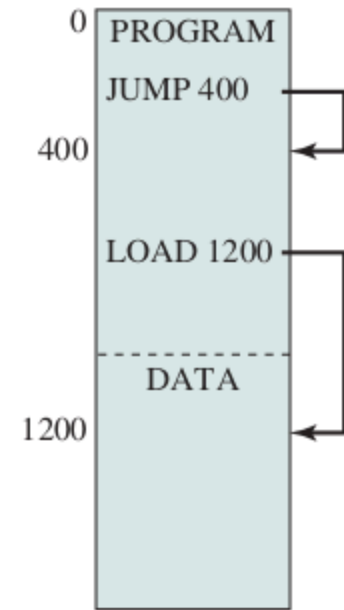
(a) Object module

Absolute
addresses



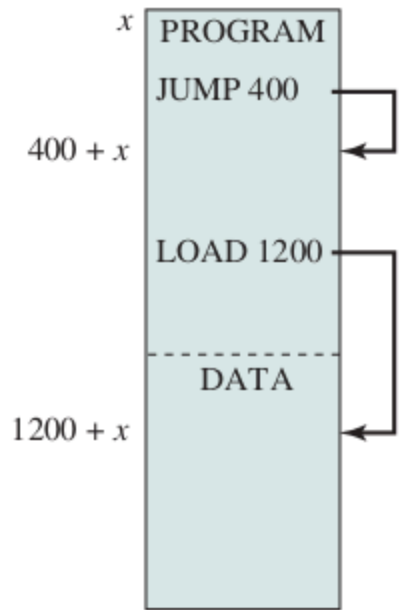
(b) Absolute load module

Relative
addresses



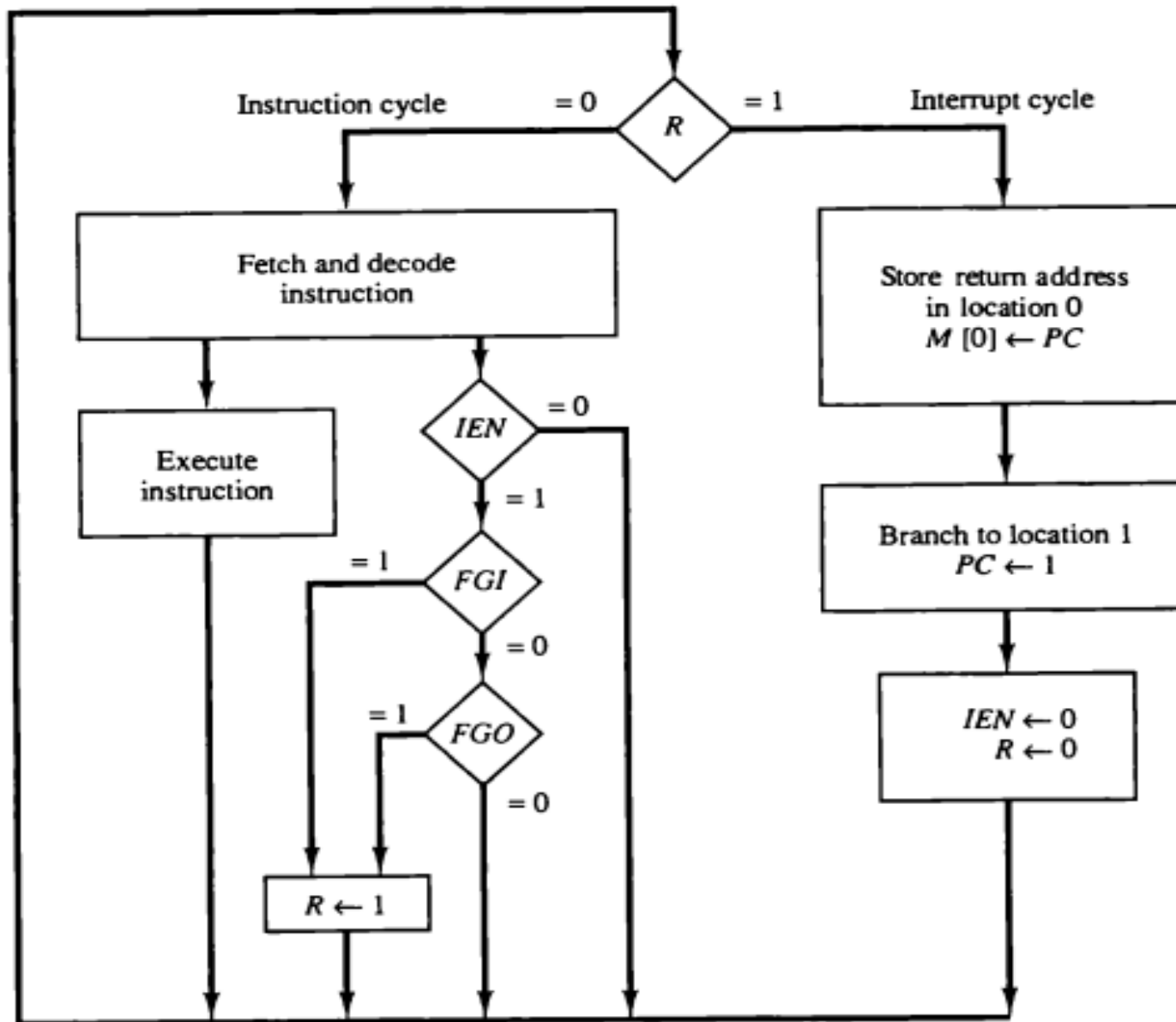
(c) Relative load module

Main memory
addresses



(d) Relative load module
loaded into main memory
starting at location x

YIC80 - Interrupt



Interrupt-Driven Program

1		BUN	MAIN
2		BUN	ISR
3			
4	MAIN,	LDA	ZERO
5		STA	COUNT
6		ION	
7			
8	WORK,	LDA	COUNT
9		INC	
10		STA	COUNT
11		BUN	WORK
12			
13	ISR,	STA	SAVE
14		BSA	IO
15		ION	
16		LDA	SAVE
17		BUN	0 I

18	IO,	HEX	0
19		SKI	
20		BUN	OUTPUT
21		INP	
22		STA	BUFFER
23		BUN	IO I
24	OUTPUT,	SKO	
25		BUN	IO I
26		OUT	
27		BUN	IO I
28			
29	ZERO,	DEC	0
30	COUNT,	DEC	0
31	SAVE,	DEC	0
32	BUFFER,	DEC	0
33		END	

Loader with interrupt (bootstrap)

Advanced: Buffered Input with BSA Subroutines

src/in/Interrupt_Driven_Program_with_BSA_Subroutines_Advanced_with_buffer.asm

src/in/Interrupt_Driven_Program_with_BSA_Subroutines_Advanced_with_buffer_comments.asm

Enhanced bootstrap loader with error checking

src/in/Bootstrap_Loader_Program_More_Robust_Version_with_Error_Checking.asm

Relocating Bootstrap Loader with Base Register

src/in/loader4_base_register_comments.asm

Simple Bootstrap Loader with Absolute Addressing

src/in/loader7_interrupt_Simple_Bootstrap_Loader_with_Absolute_Addresssing.asm

src/in/loader7_interrupt_Simple_Bootstrap_Loader_with_Absolute_Addresssing_comments.asm

src/in/loader7_user_program_comments.asm

src/in/loader7_user_program.asm

Uses interrupt-driven I/O instead of polling

src/in/loader10_interrupt.asm

src/in/loader10_interrupt_comments.asm

Interrupt-driven program that gets loaded

src/in/loader10_loaded_program.asm

src/in/loader10_loaded_program_comments.asm

```

1  ORG 0
2  BUN LOADER
3  BUN ISR
4  ORG 100
5  LOADER, LDA ZERO
6  STA BYTES_LOADED
7  STA PROG_SIZE
8  STA INP_BUF
9  W4SIZE, LDA INP_BUF
10  SZA
11  BUN STORE_SIZE
12  BUN W4SIZE
13  STORE_SIZE,
14  LDA INP_BUF
15  STA PROG_SIZE
16  STA INP_BUF
17  BSA LOAD_PROGRAM
18  ION
19  BSA USER_PROG I
20  BUN LOADER
21
22  ISR, STA SAVE_AC
23  SKI
24  BUN CHECK_OUTPUT
25  INP
26  STA INP_BUF
27  BUN ISR_EXIT
28  CHECK_OUTPUT,
29  SKO

```

```

30  BUN ISR_EXIT
31  LDA OUTPUT_PENDING
32  SZA
33  BUN HAS_PENDING_OUTPUT
34  BUN ISR_EXIT
35  HAS_PENDING_OUTPUT,
36  LDA OUTPUT_DATA
37  OUT
38  LDA ZERO
39  STA OUTPUT_PENDING
40  ISR_EXIT,
41  LDA SAVE_AC
42  BUN INDIRECT
43  SYS_WRITE_HANDLER, HEX 0
44  STA OUTPUT_DATA
45  LDA ONE
46  STA OUTPUT_PENDING
47  SKO
48  BUN DEVICE_READY
49  BUN SYS_WRITE_RETURN
50  DEVICE_READY,
51  LDA OUTPUT_DATA
52  OUT
53  LDA ZERO
54  STA OUTPUT_PENDING
55  SYS_WRITE_RETURN,
56  BUN SYS_WRITE_HANDLER I
57
58  LOAD_PROGRAM, HEX 0
59  LDA ZERO

```

```

60  STA LOAD_INDEX
61  LOAD_LOOP,
62  LDA LOAD_INDEX
63  SUB PROG_SIZE
64  SPA
65  BUN WAIT_FOR_BYTE
66  BUN LOAD_DONE
67  WAIT_FOR_BYTE,
68  LDA INP_BUF
69  SZA
70  BUN STORE_BYTE
71  BUN WAIT_FOR_BYTE
72  STORE_BYTE,
73  LDA LOAD_INDEX
74  ADD USER_PROG
75  STA STORE_ADDR
76  LDA INP_BUF
77  STA STORE_ADDR I
78  STA INP_BUF
79  LDA LOAD_INDEX
80  INC
81  STA LOAD_INDEX
82  BUN LOAD_LOOP
83  LOAD_DONE,
84  BUN LOAD_PROGRAM I
85
86  USER_PROG, HEX 500
87  ZERO, DEC 0
88  ONE, DEC 1
89  PROG_SIZE, DEC 0
90  LOAD_INDEX, DEC 0

```

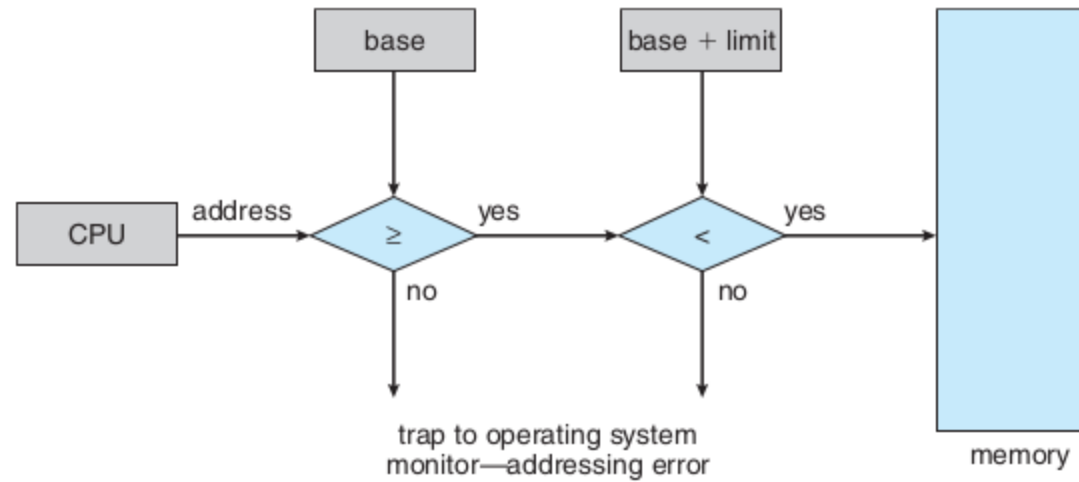
Interrupt and Relative Address Problem

- Normal execution
- Interrupt time
- ISR needs
 1. predictable, fixed addresses
 2. resources across all processes
 3. Safety

Solution Dual Address Spaces

1. User mode (MOD 1)
2. Kernel mode (MOD 0)

YIC90 - Memory and CPU Protection

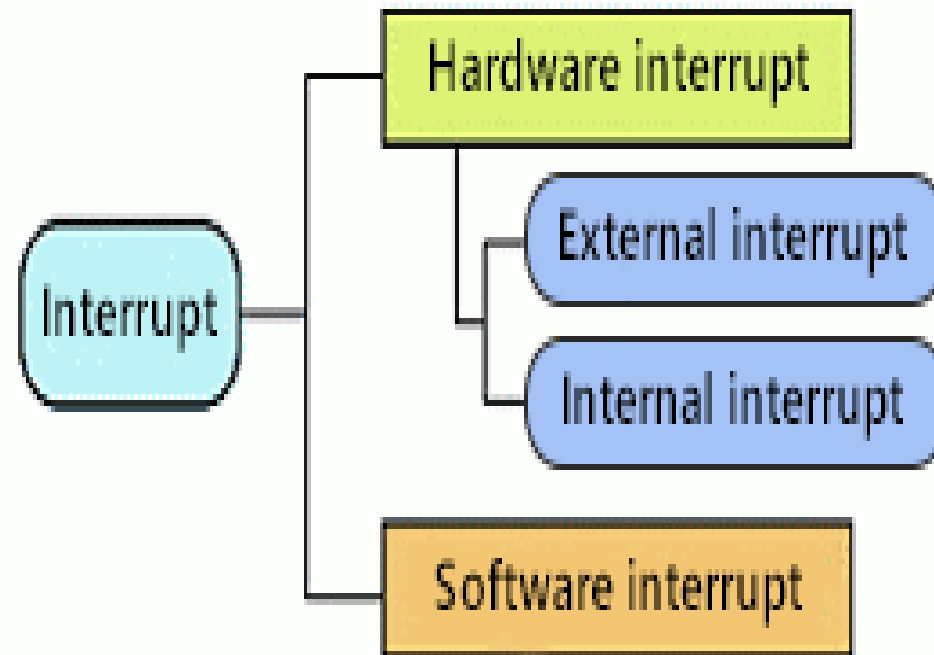


1. SKT like SKI and SKO
2. System Call ?
3. Change registers by the running process

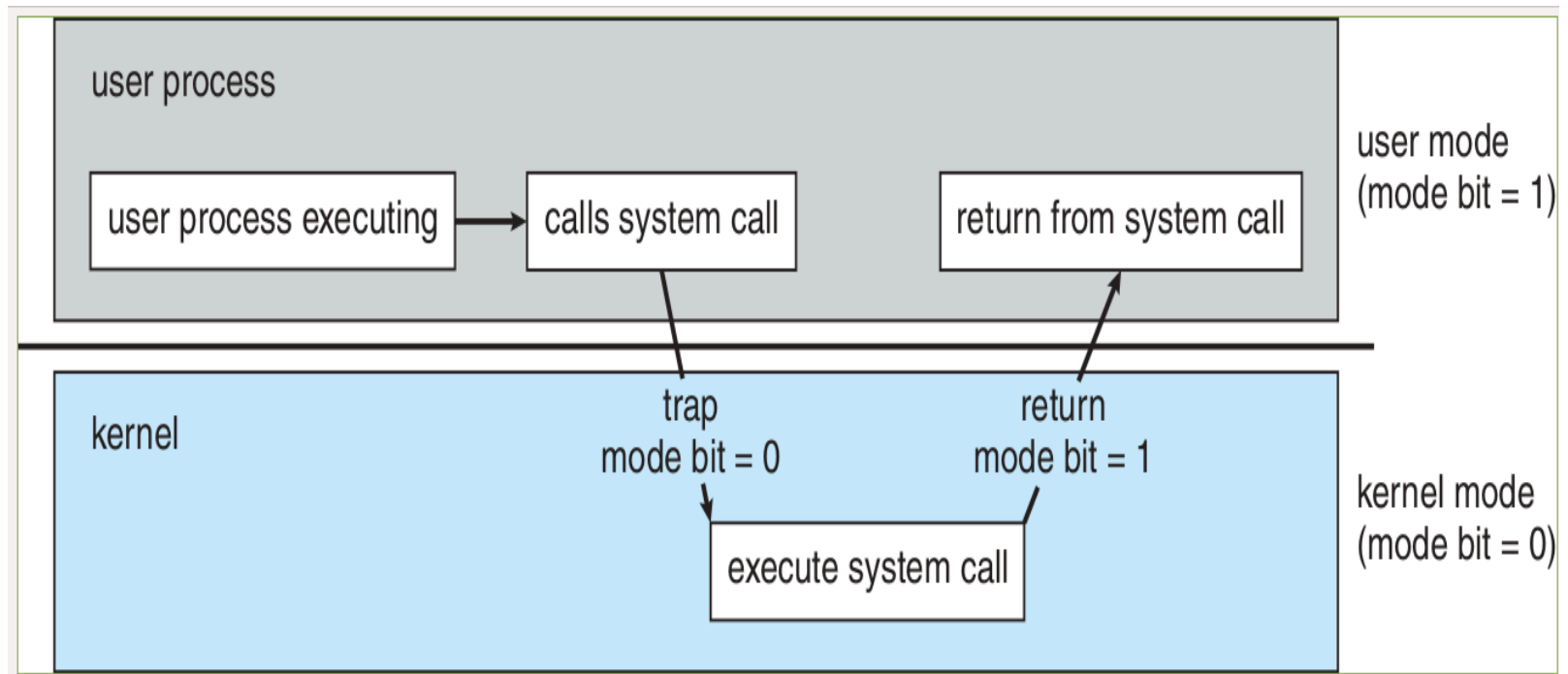
Software Interrupt

ISR,	STA	SAVE
	BSA	IO
	ION	
	LDA	SAVE
	BUN	0 I
IO,	HEX	0
	SKI	
	BUN	OUTPUT
	INP	
	STA	BUFFER
	BUN	IO I
OUTPUT,	SKO	
	BUN	TRAP
	OUT	
TRAP,	SKT	
	BUN	IO I
	BUN	100

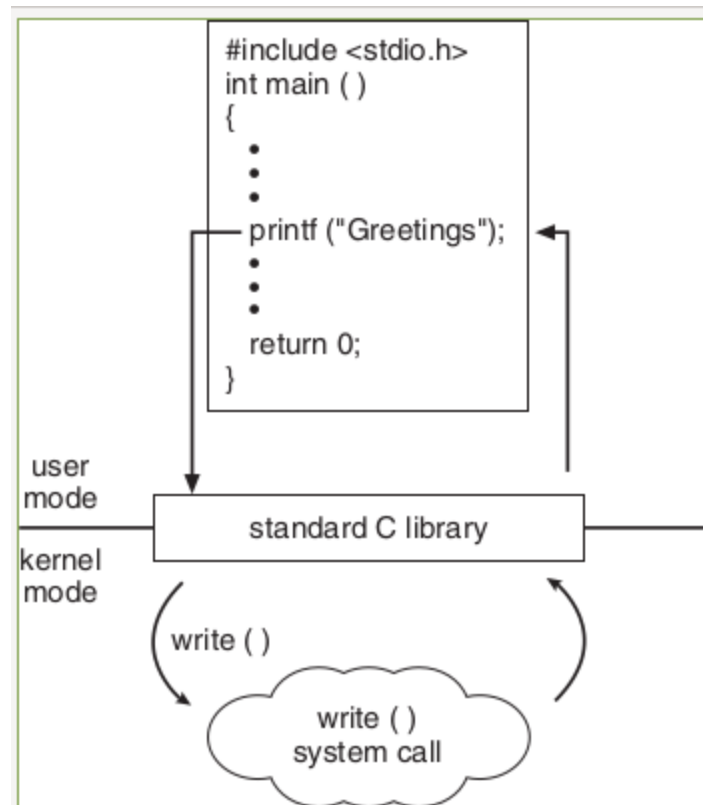
```
mov ah, 0x0e
; function number = 0Eh
; : Display Character
mov al, '!'
; AL = code of character
; to display
int 0x10
; call INT 10h,
; BIOS video service
```



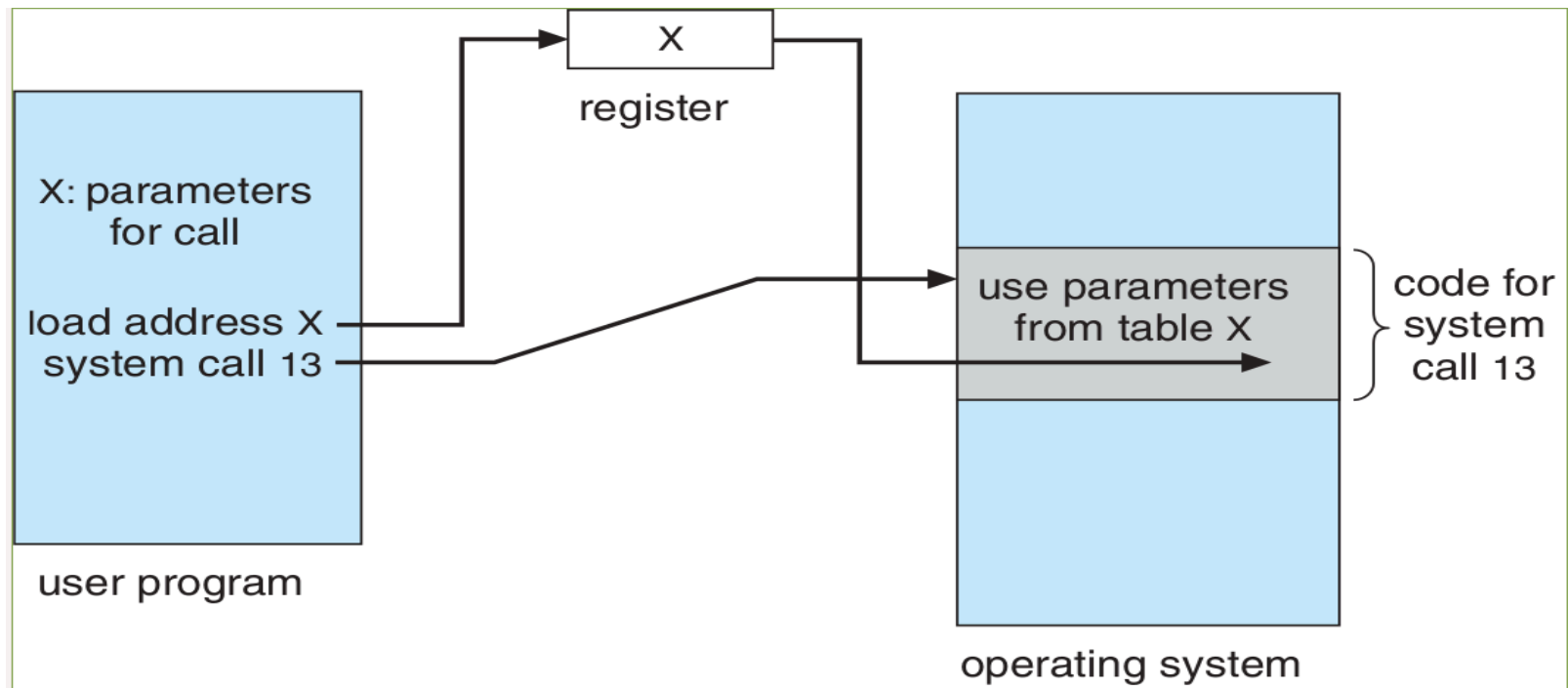
System Call



C System Call



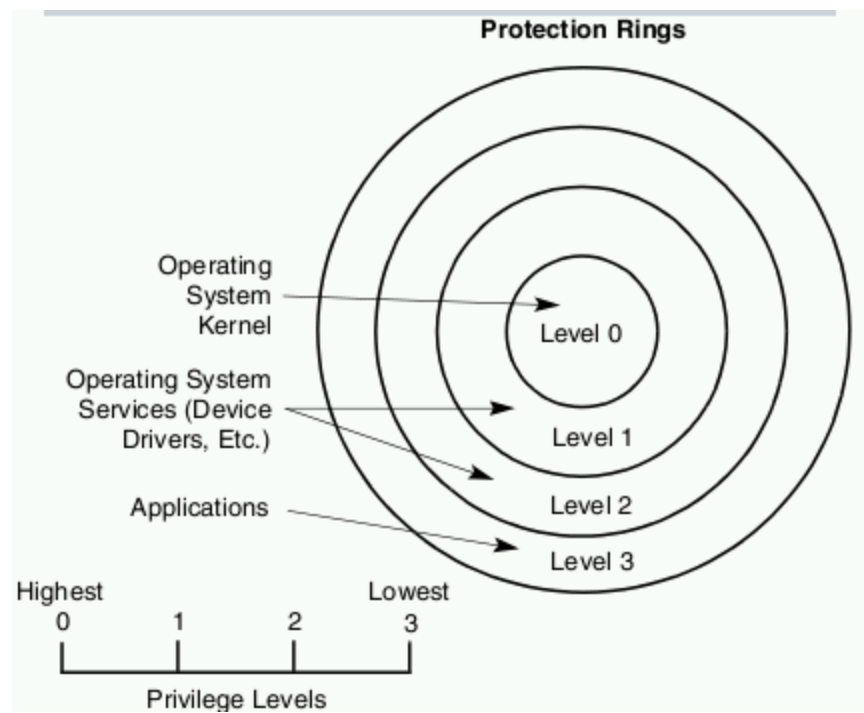
Simple Parameters



- kernel mode
- user mode

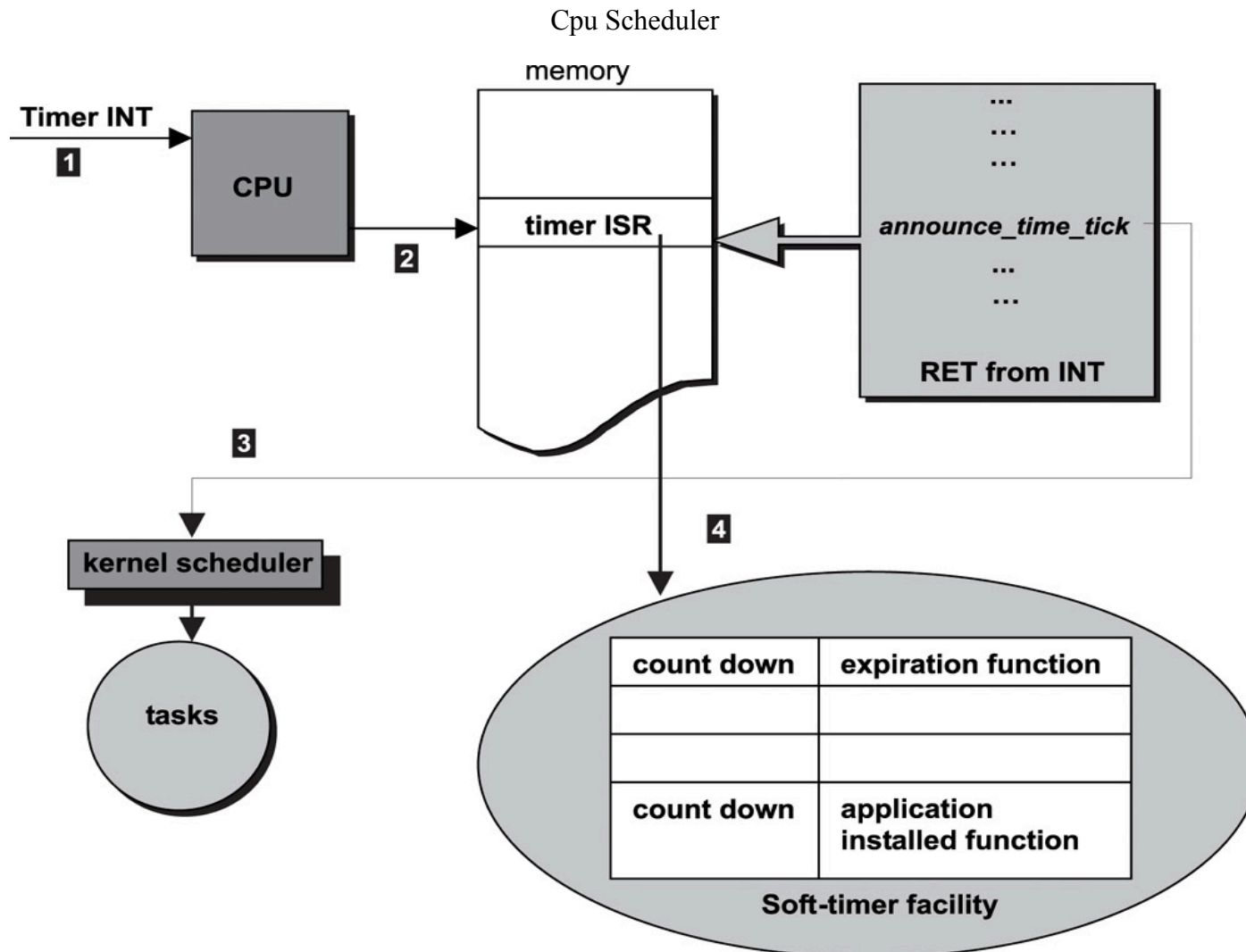
Pentium 4 (ESCR)





CPU protection

Timer interrupt



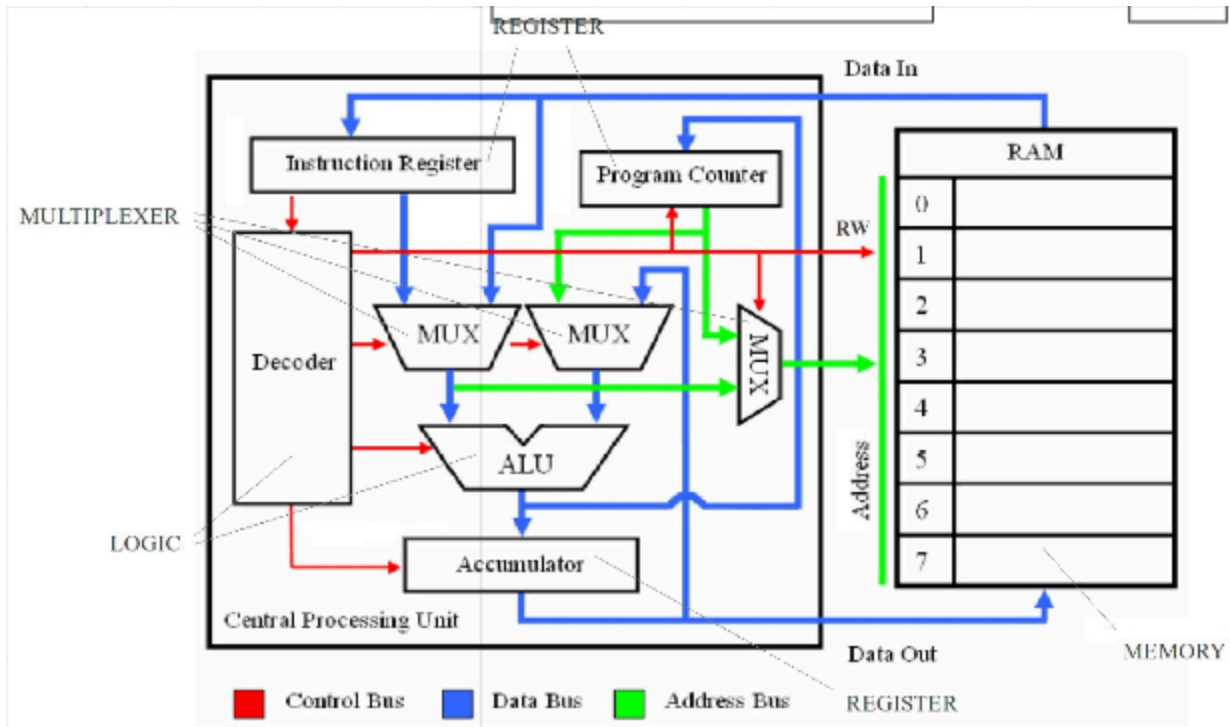
YIC110 - Multiprogramming

Function call

- cons of BSA
 - No recursion
 - No explicit data transfer

Stack From end

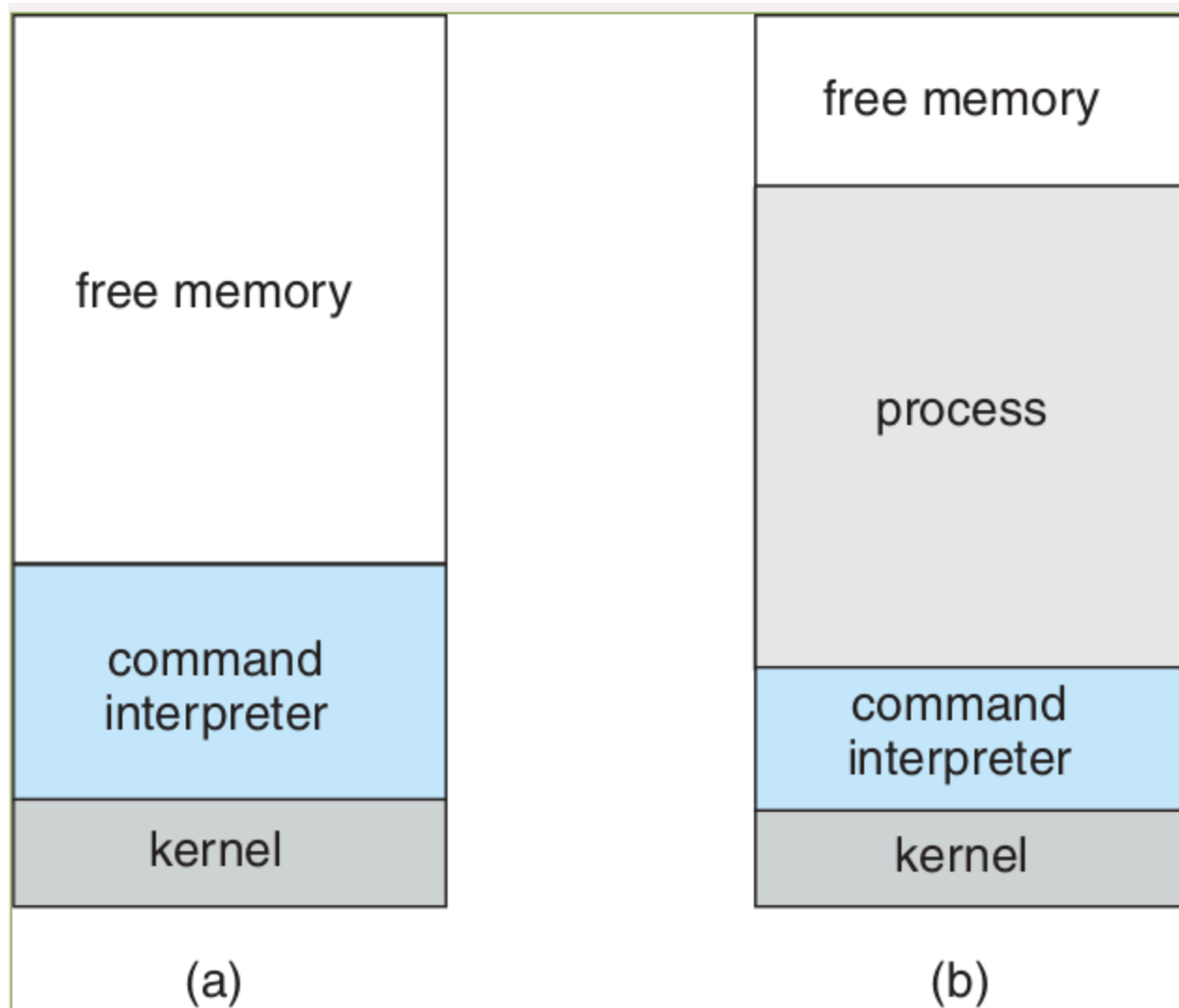
Call, Ret



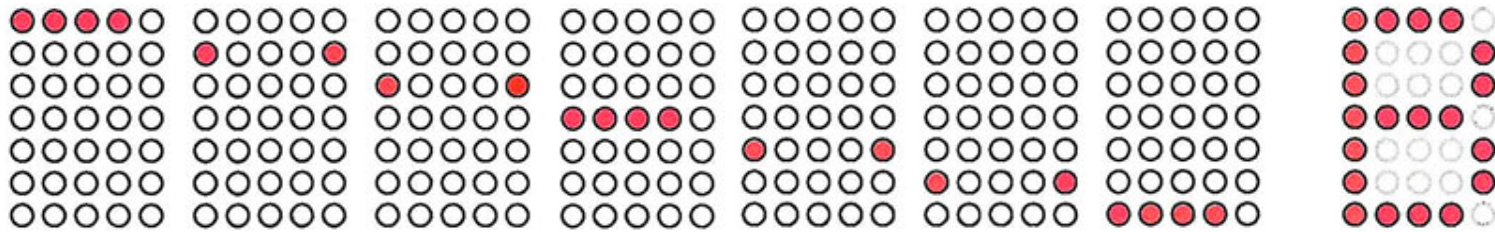
- Assembly Slides

YIC120 - Adding Keyboard & Disk

- terminal (command prompt)
- batch system
- interactive system

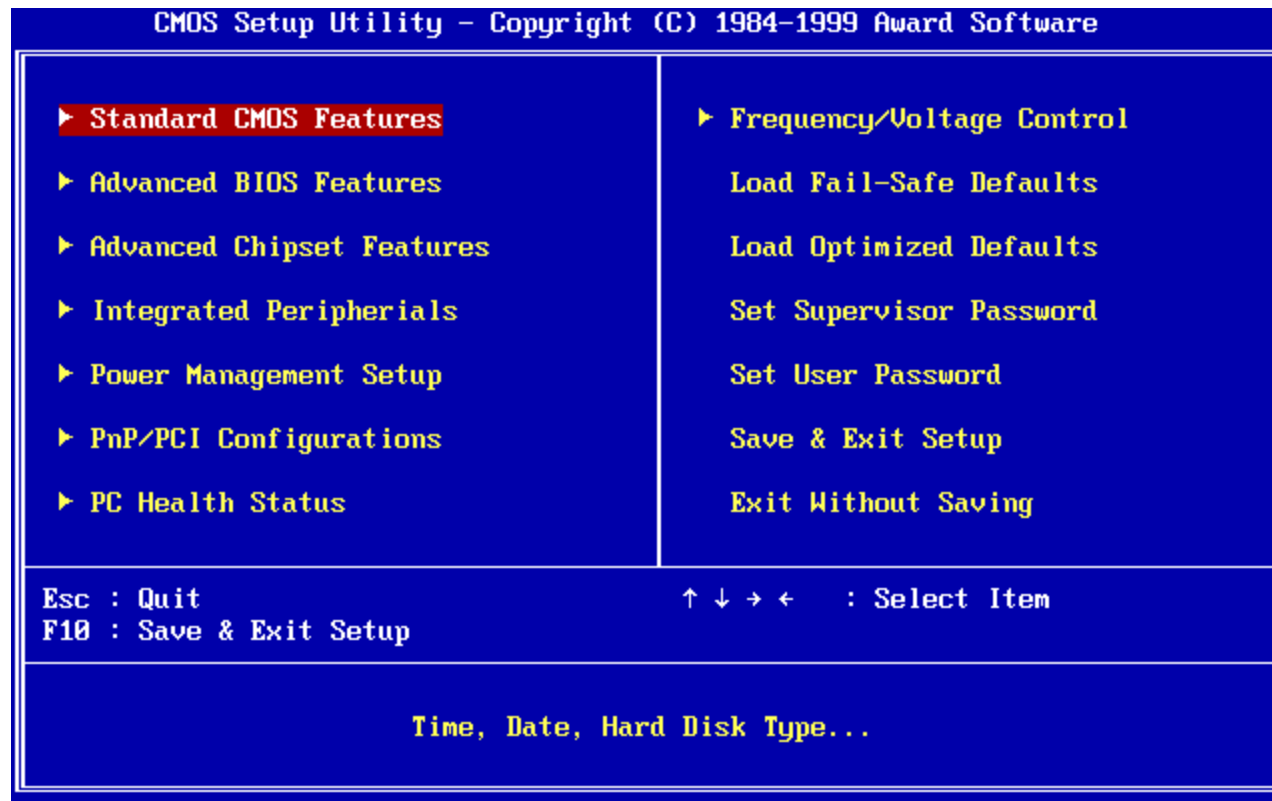


When a controller rapidly turns on LEDs in one row at a time



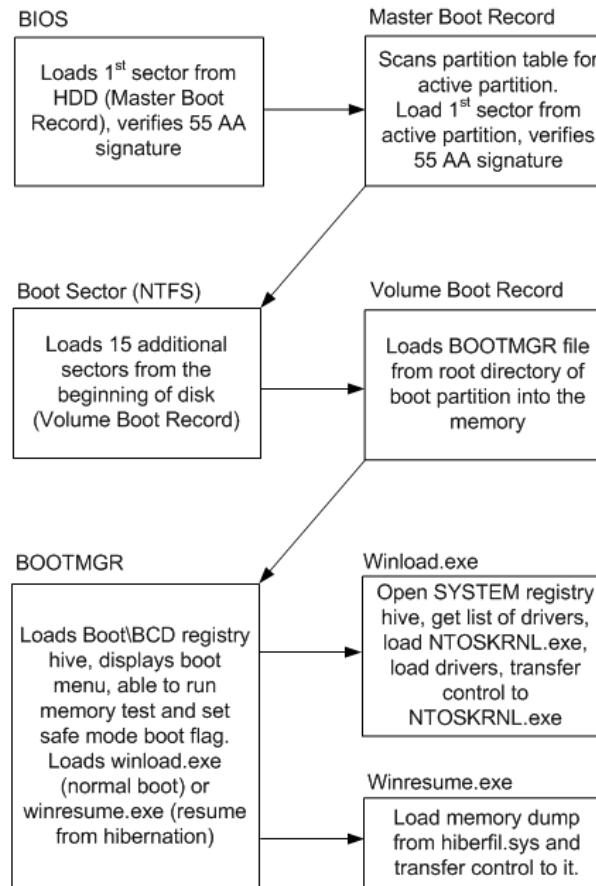
<https://www.nutsvolts.com/magazine/article/create-an-led-sign-controller>

BIOS



Main	Security	System Configuration	Exit
System Time		[22:58:20]	
System Date		[07/26/2016]	
Product Name		HP ENVY x360 Convertible 13-y0XX	
Product Number		YODSKU1#ABA	
System Board ID		82B7	
Born On Date		07/25/2016	
Processor Type		Intel(R) Core(TM) i7-750U CPU @	
		2.70GHz	
Total Memory		16 GB	
BIOS Version		B.09	
BIOS Vendor		American Megatrends	
Serial Number		36444335-3432-5842-3053-	
UUID Number		535834324435	
		PYD4F018J30019	
System Board CT Number		Win10	
Factory installed OS		24852 05/26/2016	
Primary Battery SN			
► System Log			
Build ID		16WW3DST6A#SABA#DABA	
Feature Byte		3K6b 7K7N 7WaB apaq asau awbV	
		bhbz cbdU dXdp dqew .E5	
1	Help	↑↓ Select Item	F5/F6 Change Values

Boot sequence



```
Ubuntu 8.04, kernel 2.6.24-16-generic
Ubuntu 8.04, kernel 2.6.24-16-generic (recovery mode)
Ubuntu 8.04, memtest86+
Other operating systems:
Windows Vista/Longhorn (loader)
```

Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, 'e' to edit the
commands before booting, or 'c' for a command-line.

The highlighted entry will be booted automatically in 4 seconds.

POWER ON COMPUTER



BIOS PERFORMS POST



DISK



LOADING OF OS



RAM

- [IEEE Std 1275 1994 Standard for boot initialization](#)
- https://openfirmware.info/Welcome_to_OpenBIOS
- <https://github.com/openbios>
- <https://github.com/openbios/openbios>

Context Switch

END

