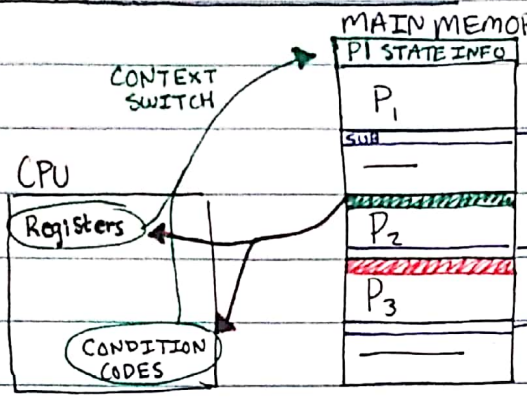
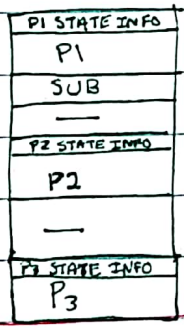
SUBROUTINE

REENTRANT — MULTIPROCESSING



This might be too big to store 3 times in memory. So INSTEAD, CAN we JUST STORE IT ONCE and share it between the processes. It needs to be "reentrant" for this to work



Need to Avoid data From one program staying in the Sub, cause it could end up being used by the other program

4!

RECURSIVE

FACTORIAL EXAMPLE

~~R1: 1*2=2~~
~~R1: 2*3=6~~
R1: #6*4=24

FACT(N) pass in and out by value

VAR TEMP:INT

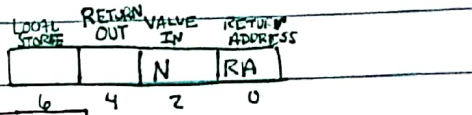
BEGIN

IF N=1 return 1

ELSE TMP:= N-1

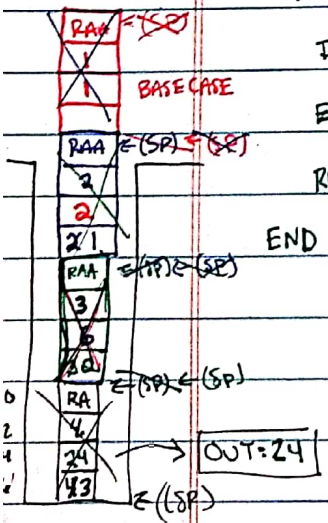
RETURN N*FACT(TMP)

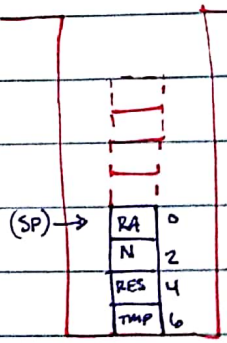
END



SAVE THE REGISTERS →

```
DRIVER: SUB #8.,(SP)
  MOV N,2(SP)
  MOV #RA,0(SP)
  JMP FACT
RA: MOV 4(SP), OUT
  ADD #8., SP
;RESTORE REGISTERS
```






```
FACT: CMP #1., 2(SP)
  BNE BASIS
  MOV 2(SP), 6(SP)
  SUB #1., 6(SP) ; or DEC6(SP)
MOV SP,R1 → SUB #8, SP
  MOV 6(OLDSP), 2(NEWSP) ; USE REGISTER TO
  R1          SP           store old SP
  MOV #RRA, 0(SP)
  JMP FACT
RAA: MOV 4(SP), R1
```

```
BASIS: MOV #1, 4(SP)
  JMP @0(SP)
```

```
ADD #8., SP
MUL 2(SP), R1
MOV R1, 4(SP)
JMP @0(SP)
```

Initialize a stack

STACK: .BLKW 1000.

START: MOV #2000, SP or MOV #START, SP ← only if nothing is in between

Boolean Data Type (TRUE/FALSE)

AND, OR, NOT, XOR - SUB, BNE

PDP-11 is word and byte addressable but not bit addressable

BIC src, dst BIT CLEAR (BYTE)

BICB src, dst

(dst) ← ~(src) ∧ (dst)

N: MOB Z: if 0 V: Always 0 C: unchanged

BICB X, Y

Y 0000 0001

X 0000 0001

~X ∧ Y

1111 0110 ∧ 0000 0001 = 0000 0000

0000 0000 → (dst) clears any bit in Y where the corresponding bit in X is 1

Packing - Storing multiple values in one word or byte

Y 1111 0000 X 0101 0101

~X 1010 1010

(dst) ← 1000 0000

BIS src, dst BIT SET (BYTE)

BISC src, dst

O(N) divide algorithm

Due week from Tues Nov. 5th

Non-Restoring Division

Do another example by hand due Thursday 7th

X = dividend
Y = divisor
Q = quotient -> $q_{15} q_{14} \ldots q_1 q_0$ {The 16 bit quotient is made up of bits $q_{15} q_{14} \ldots q_1 q_0$ }
R = remainder
N = number of bits in word = 16.

1. Concatenate N zeroes to X getting D, i.e., make it **double precision**.
   Set i to N-1

2. Subtract $2^i * Y$ from D call this result {note: this requires **double precision** subtraction and **double precision** $2^i * Y$}

3. If the result is negative $q_i = 0$ {bit i of quotient gets set to 0}
   If the result is non-negative $q_i = 1$
   i := i - 1
   If i = -1 then goto step 5 else goto step 4

4. If result is negative add $2^i * Y$ to result to form new D {again **double precision** needed here}
   else subtract $2^i * Y$ from result to form new D
   Goto step 3

5. If the result is non-negative result is remainder.
   If the result is negative add Y to result to get remainder

---

Work an example by hand. Use 6 bit integers for this example. This must be done in **binary**,

For this assignment you **may not** use MUL or DIV

```
X:      .BLKW     1       ;IN - 2's compl >= 0  Dividend
Y:      .BLKW     1       ;IN - 2's compl >= 0  Divisor
Q:      .BLKW     1       ;OUT - X div Y or (-32768. as error)  Quo
R:      .BLKW     1       ;OUT - X mod Y or (-32768. as error) Remainder
```

Non zero positive integers only

Don't even have to test negative

rexp (m,n)
if n=0 then return 1
else if even(n) then return (square(rexp(m, n/2))
else return (n * rexp(m,n-1)

0	Return Address
2	Place for result
4	N
6	m

MPOWN	112	~~2102~~	2112
m	~~110~~	~~2110~~	2110
N	~~106~~	~~2106~~	2106
STACK	114	~~2000~~	2000

```
rexp:  TST N
       BEQ z ; return 0
       MOV N, R1
       CLR R0
       DIV R0, #2.
       TST R1   ; Divide N by 2 and look at the remainder
       BEQ EVEN; if n is even (r=0)
       ;if N is odd

EVEN:
z:
```

MPOWN 2104
m 2102
~~MPOWN 2~~
N 2100
~~2004~~
Start 2000
MPOWN 2112
m 2110
N 2106
START 2000

Locality | Spacial | Temporal
N Δt | likely to access things near N | likely to access N again

BICB
BICB src, dst
BIC src, dst
(dst) ← ~(src) ∧ (dst)
X 1111 0000 → 0000 1111 BICB X Y
Y 1010 1010 → ∧ 1010 1010
0000 1010 → dst

Write Bits

BISB src, dst
BIS src, dst
(dst) ← (src) ∨ (dst)
1111 0000
1010 1010
1111 1010 → (dst)

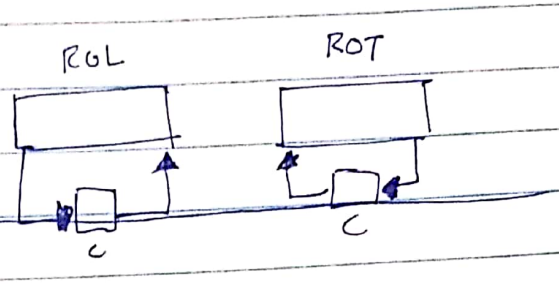
Logic

Neg → 2's comp
Com dst
ComB dst → 1's comp logical negation (NOT)

BIT src, dst (Like TST but for bits)
BITB src, dst (src) ∧ (dst)
N HOB Z - if all 0s V - 0 C - Not Changed

Rotate

ROL (B) dst
ROR (B) dst



Arithm

ASH src, dst

ASC src, dst

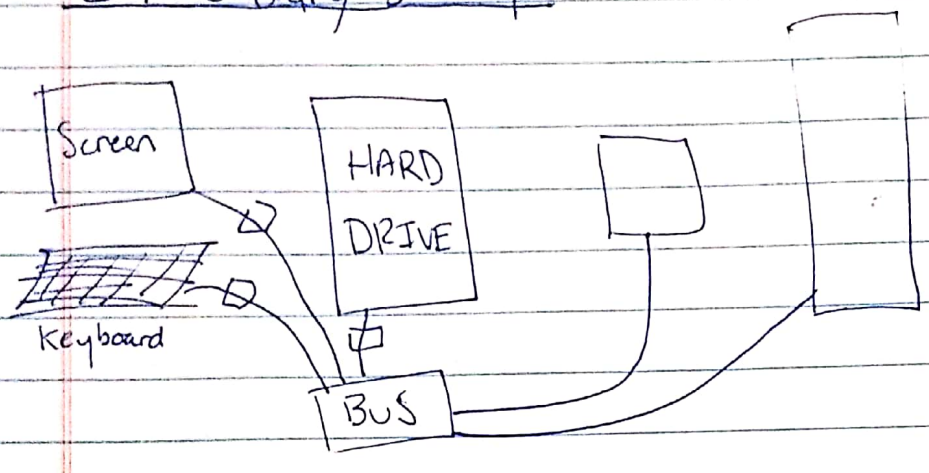
Arithmatic Shift → Java shift (multiple shifts)

ASC → double precision

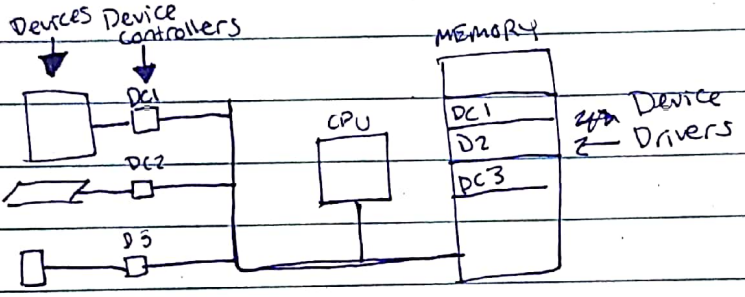
ASC #-1, dst → left shift by 1

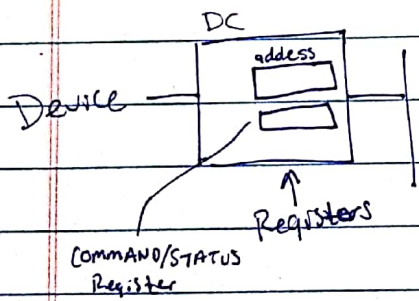
ASC #3, dst → right shift by 3

IO/Secondary Storage



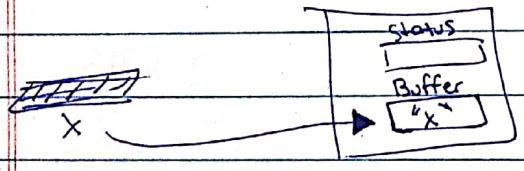
Some BUS(ES) to connect it all together





Since the registers are only one way (read or write) one address is good enough to map a pair of registers

Additional registers to store information



Asynchronis System

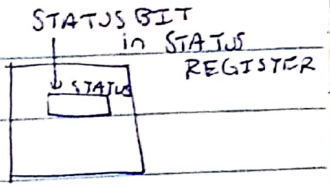
Driver has to take data out of buffer register before the next key is hit

So Device Driver monitors status register that says "hey there's a character"

Device Drivers

Programmed I/O, Polling

STATUS BIt, character ready to retrieve
, & device ready for command

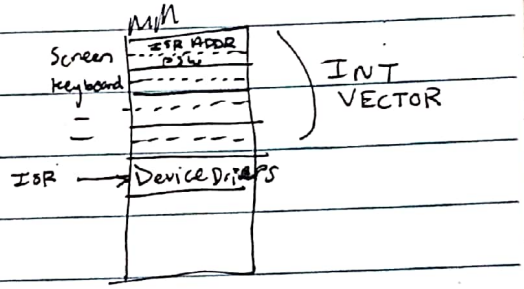


Busy Wait Just ask the status over and over until something happens
- Uses all of the CPU waiting for input
  - ↳ still used sometimes when cpu has nothing else to do when waiting for input
  - → low cost driver implementation

Interrupts (Vector Driven)

Interrupt Driven Device Driver

1. Complete current instruction



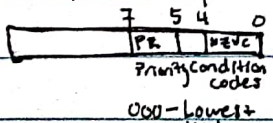
2. Save (PC) and (PSW) so we can come back
   - save on SP's stack

Hardware calls interrupt service routine, Not allowed to use most registers to save on state information

3. Goto interrupt vector, put stored addr of ISR → (PC)
   PSW bits → (PSW)

4. ~~RTS~~ RTI → Return from interrupt
   from SP's stack restore (PC) and (PSW)

Not all interrupts are equal. So we need to add priority

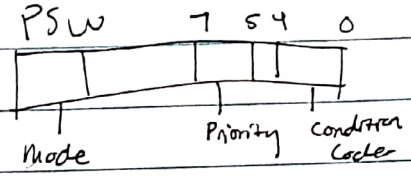


Masked Interrupt

But what stops Prog 1 from accessing or manipulating Prog 2 or using the Driver Controllers being used by other processors.

Only one CPU, can't run OS and prog at the same time so we need hardware

Primitive - Bound Registers



Tier of Privileges

User Programs Can't use the halt command cause that would kill the OS and other programs too

Hard drive uses main memory to buffer the blocks of data
This requires another processor built into the Device Controller
DMA DC → Direct Memory Access Device Controller
- Extra register to hold main memory buffer address
- extra register to hold read block size is common
- cycle stealing - hard drive gets priority main memory access due to its time sensitive nature

1.067 mm

[2.5 amps
[1.25 amps
[1.25 amps
[2.5 amps

7.5

$>R \xrightarrow{\prime} \#R_{c,d,\square} \xrightarrow{c} \%$

D
↓
y

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	0
1011	1
1100	2
1101	3
1110	4
1111	5

$S \rightarrow \%T$

$T \rightarrow ABTAB$

$T \rightarrow \#$

① Generate unordered

$\%A^nB^n\#A^nB^n$

$(\%(AB)^n\#(AB)^n)$

$BA \rightarrow AB$

② order it

$\%A \rightarrow a$

$aA \rightarrow aa$

$aB \rightarrow ab$

$bB \rightarrow bb$

$\#A \rightarrow a$

$\%\# \rightarrow \varepsilon$

③ convert A to a and B to b

Gen aabbaabb

$S \Rightarrow \%T$

$\%ABTAB$

$\%ABABTABAB$

$\%ABAB\#ABAB$ ①

$\%AABB\#AABB$ ②

$aABB\#aABB$

$aaBBaaBB$

$aabBaabB$

$aabbaabb$ ③

Gen ε

$S \Rightarrow \%T$

$\%\#$

ε

$L = \{a^{2^n} : n \geq 0\}$

$S \Rightarrow \#a\%$

$\# \Rightarrow \#D$

$Da \Rightarrow aaD$

$D\% \Rightarrow \%$

$\# \Rightarrow \varepsilon$

$\% \Rightarrow \varepsilon$

Compute Compliment

$S10110112S$

$S1 \Rightarrow 0S$

$S0 \Rightarrow 1S$

$SS \Rightarrow \varepsilon$

Increment

$0S \rightarrow T1$
$1S \rightarrow S0$ — Seek first zero

$SS \rightarrow 1$ — There is no zero

$0T \rightarrow T0$
$1T \rightarrow T1$
$ST \rightarrow \varepsilon$ — Flip and carry

Given grammar or turing machine
↳ run input, is it accepting

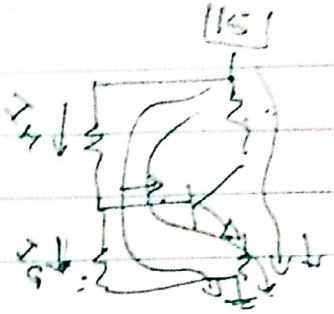
Build turring machine to accept a given language

Classify Language (Context Free, Regular, Not Context Free)

- Context Free → PDA
- Regular → FSM, Regular Expression
- Not Context Free → context free pumping Lemma

Not Regular → Pumping Lemma

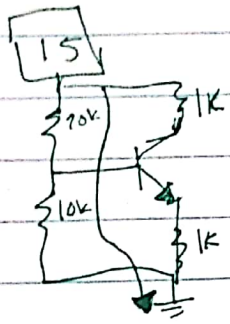
Context free intersect regular → context free



~~$15 - 20k I_H = 0$~~

~~$I_H = \frac{15}{20000} = 750\mu A$~~

$15 - 20k I_H - .795 - 1k I_E = 0$



$15 - I_H 20k - I_G 10k = 0$

$15 - (I_B + I_G)20k - 10k(I_G) = 0$

$-20k I_B - 20k I_G - 10k I_G = -15$

$-20k I_B - 30k I_G = -15$

$-20k I_B - 20k I_G - V_{CE} - 1k I_E = -15$

$-20k I_G - 1k I_E - V_{CE} = 4.76$

$15 - (I_B + I_G)20k - V_{CE} - 1k I_E = 0$

$-20k I_B - 20k I_G - 1k I_E = -15 + .795 = -14.205$

$-1k I_E - 1k I_B - I_E - V_{CE} = 0$

$15 - I_C 1k - V_{CE} - 1k I_E = 0$

I_G	I_E	V_{CE}	
0	-2K	-1	14.469
-20K	-1K	-1	4.76
-20k	-30k	0	-15

$-1k(I_E - I_B) - 1k(I_E) = -14.205$

$-1K I_B - 2k I_E = -14.205$

-20 -30 0

CS 435 Final Exam Topics

will be told context and regularity
prove it

1. Regular languages
   (a) construct a deterministic FSM
   (b) construct a nondeterministic FSM
   (c) prove a language is not regular

easier form of (b)

pumping lemma

2. Context-free languages
   (a) construct a PDA (1 stack)
   (b) write a context-free grammar
   (c) prove a language is not context-free

$S \to x$ (cf)
$Sx \to xx$ (not cf)

pumping Lemma

3. Decidable languages
   (a) construct a TM
   (b) write an unrestricted grammar

c) $\{x\#y : x, y \in \{0,1\}^* \text{ and } x \neq y\}$

Context Free not regular. We can create a context free grammar or Push Down Automata that accepts this language

$S \rightarrow TH \mid RT$

$H \rightarrow TL \mid A$

$R \rightarrow RT \mid A$

$T \rightarrow 0 \mid 1$

$A \rightarrow TAT \mid \#$

use pumping lemma to show it is not regular

9/10

Theory of Comp — Adam Stammer

13 - Non Context Free Languages $|a,b,c$

1. Regular? Context Free? Prove your answer

a) $L = \{xy : x,y \in \{a,b\}^* \text{ and } |x| = |y|\}$

Regular. We could say that the above language is combos of a and b with an even length. This is something we can track with a ~~regex~~ simple FSM.

$w = \{aa \cup ab \cup bb \cup ba\}^*$

b) $\{(ab)^{n_1} a^{n_2} b^{n_3} : n > 0\}$ $\quad (ab)^k\ a^k\ b^k$ (① ② ③)

Not context free.

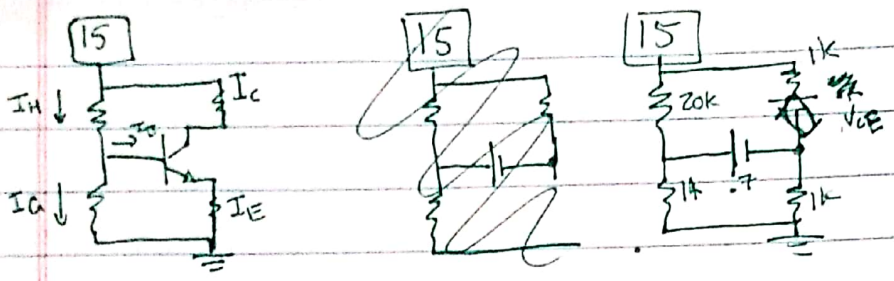
(1,1) if pumped up when odd it puts the letters out of order
if pumped up when even we don't have enough a's or b's in the next region to match

(2,2) pump up once and |region 2| > |region 3|

(3,3) pump up once and |region 2| < |region 3|

(1,3) This would make |vxy| greater than k

(1,2) Depending on where they fall in ab pumping up once would either put the region 1 letters out of order or put more a's or b's in region 1 than regions 2 and 3 respectively



$15 - 20k I_H - 10k I_G = 0$

$-20k I_H - 10k I_G = -15$

$15 - 20k(I_H) - .7 - 1k I_E = 0$

$-20k I_H - 1k I_E = -14.3$

$15 - 1k I_C - V_{CE} - 1k I_E = 0$

$-1k I_E - 1k I_C = -14.8$

$(I_H - I_G) + I_C = I_E$

$I_H - I_G + I_C - I_E = 0$

I_E, I_G, I_H, I_C

A				b
I_H	I_G	I_C	I_E	
-20k	-10k	0	0	-15
-20k	0	0	-1k	-14.3
0	0	-1k	-1k	-15
1	-1	1	-1	0

$A \backslash b =$

351.16 μA
797.67 μA
7.72326 mA
7.27674 mA

Sequential or Random Access

↳ In order

→ magnetic tape

Random Access - read in any order

Direct access - read in any order in the same amount of time

Row hammer attacks

Stack-smashing - Buffer Overflow

Heap Spray

```
STACK:   .BLKW 1000

START:   MOV #START, SP

         SUB #8., SP
         MOV N, 4(SP)
         MOV #RA, 0(SP)
         JMP SUB
RA:      MOV 6(SP), MPOWN
         ADD #8., SP
         HALT


SUB:     TST 4(SP)
         BRE Zero
         MOV 4(SP), R1
         CLR R0
         DIV #2., R0
         TST R1                ; Look at the remainder
         BRE EVEN
ODD:     SUB #8., SP
         MOV 14(SP), 6(SP)

EVEN:                                  N 4122
                                       M 4124
                                    MPOWN 4126

ZERO:    MOV #1., 2(SP)
         JMP@0(SP)
```

        - Control Computers
        - Power
            - Supplies/Connectors
            - Low / High Power
            - Turret Terminals
        - Temperature Sensors
        - Digit Display for Debugging
- Implementation
    - Zeus2
        - Existing Equipment
        - New Cards
            - 2-4 Wire DC Bridge
            - 4 Wire AC Bridge
            - PID Controller

Summury/Conclusions-

- Important take aways
- Next Steps
    - Manufacturing
    - Programming Control Software
        - Bandwidth Tests
    - OSF
    -

Abstract:

HARDWARE.astronomy: Housekeeping Box (H.aHkBox)
Here we present the HARDWARE.astronomy Housekeeping Box (H.aHkBox). The H.aHkBox is a low-cost open-source temperature monitoring and control system. It employs existing open-source devices (e.g Arduino, RaspberryPi) to reduce costs while also limiting the complexity of the development. The H.aHkBox features a chassis with a variety of capabilities, a control computer, and 10 expansion slots that can be populated by expansion cards. The first deployment of the H.aHkBox will be for the ZEUS-2 submillimeter grating spectrometer. As such the modular cards will include AC and DC excited 4-wire bridges, 2-wire bridge, and PID controller card. The system can output up to 200W, and achieve sub-millikelvin temperature sensing accuracy. Design, firmware, software and parts list will be published online allowing for other projects to adopt the system and create custom expansion cards as needed. Here we provide an overview of the project, initial layout of the chassis, electrical design, and specifications, as well as a proto-type expansion card.

Outline:

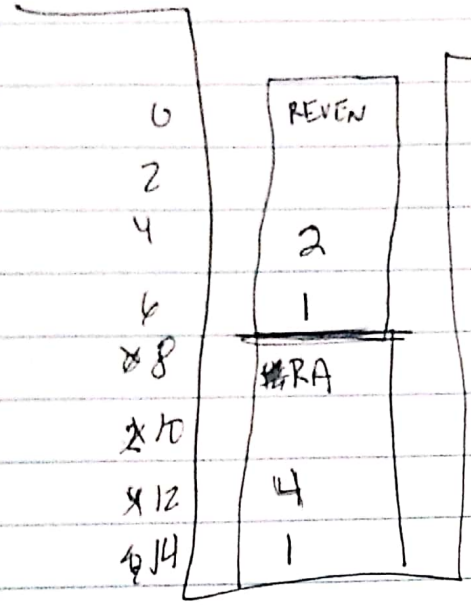
Introduction

- Hardware.Astronomy (Open Source)
- Zeus2
  - Existing Equipment
    - High Cost
    - Largely Deprecated
- FIGUREs
  - Picture of zeus2
  - Current box,
  - comparible off the shelf
-

Requirements

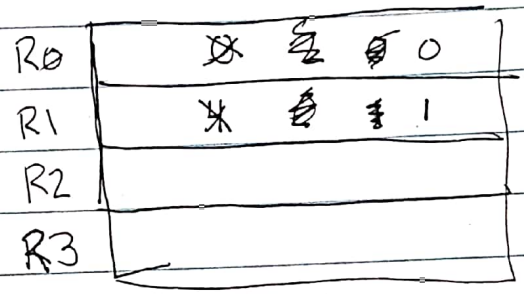
Design

- Overview
- Mechanical
- Electrical
  - Requirments
  - Expansion Cards (Slots)
    - Connectors
    - Teenzies
    - IO
      - 9 4 pin channels
      - 1 2 pin High power channel
    - Intercommunication
      - I2C x2 (Rpi, Arduino)
      - SPI Parallel (High Speed)
    - Prototyping Card
      - Breadboarding
  - Backplane Features



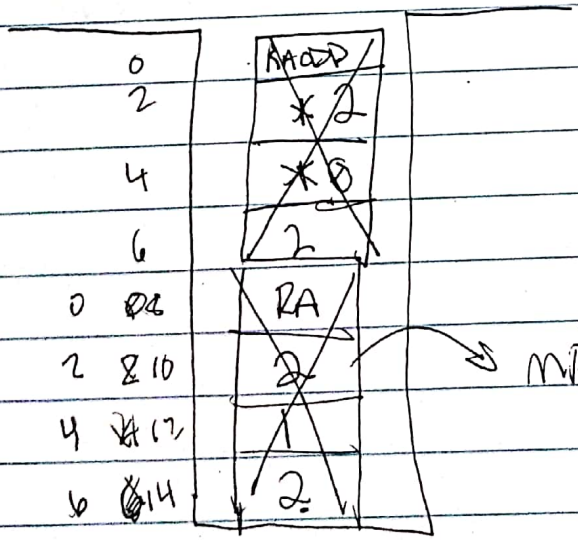
N=4
M=1

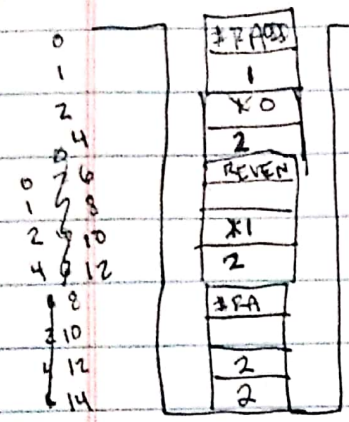
N	Z	V	C
0	0	0	0
0	1	0	0
0	0	0	0



N	Z	V	C
0	0	0	0
0	0	0	0
0	0	0	0
0	1	0	0

M=2
N=1





R0	~~4220~~
R1	~~0~~ 1
R3	

m = 2

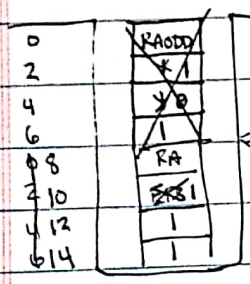
N = 2

N	Z	V	C
0	0	0	0

4226 ~~4224~~ ~~4212~~ ~~4210~~ ~~4208~~ N ~~4222~~ ~~4216~~

4230 ~~4226~~ ~~4214~~ ~~4212~~ ~~4206~~ M ~~4224~~ ~~4220~~

~~4232~~ 4230 ~~4216~~ ~~4214~~ ~~4210~~ MPOWN ~~4218~~ ~~4202~~

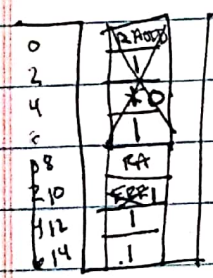


m = 1

N = 1

N	Z	V	C
0	0	0	0
0	0	0	0
0	1	0	0

R0	~~0~~
R1	* 1
R3	



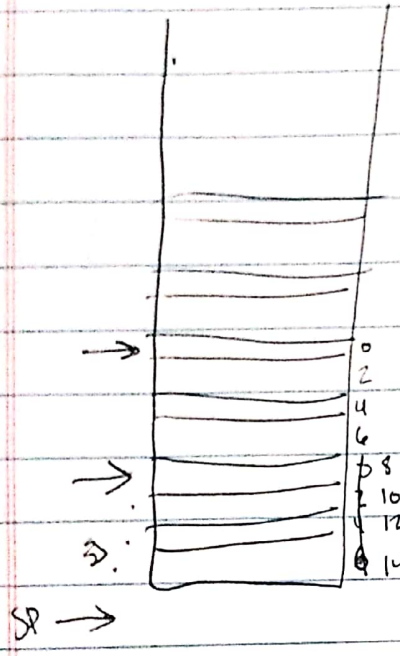
m = 1

N = 1

N	Z	V	C
0	0	0	0
0	0	0	0

R0	~~0~~
R1	* 1
R2 / R3	0 / 1

0
1



N ~~4230~~ ~~4230~~ 4226

M ~~4232~~ ~~4232~~ ~~4230~~

MPOWN ~~4234~~ ~~4234~~ 4232

N ~~4232~~ 4240

M ~~4234~~ 4242

MPOWN ~~4236~~ 4244 4

Today is Tues Nov. 12th Week From thursday 21st?

Recursive Exponentiation

Assume that m and n are two non-negative integers. The following computes m^n

```
function rexp(m, n)

        if n = 0 then return(1)
        else if even(n) then return( square( rexp(m, n/2) )
                        else return ( m*rexp(m,n-1) )
end funtion rexp
```

(3,3) (3,2) (3,1) 2

3*((3(1))

3*3^2 = 3*3*3 = 27

You need to write a **driver** program and a **recursive subroutine**.

Label the first instruction in the driver program **START:**

To run your program, address **START needs to be in the PC.**

rexp(4,1)

m*rexp(m,n-1)

4*1 = 4

Label the first instruction of the subroutine **REXP:**

Separate the driver and subroutine by a block of comments.

The driver program builds the activation record and then *calls* the subroutine.

The subroutine should implement the recursive algorithm defined above.

The driver program needs a stack for the activation records.

Create one by starting your program as follows.

```
            .TITLE    Recursive Exponentiation
            .BLKW     1000.
START:      MOV       #START,SP ;initialize stack pointer
                                ;to address START
```

go start. Not go 0.

Be sure you start executing the program at address **START**, not zero.

Activation Record

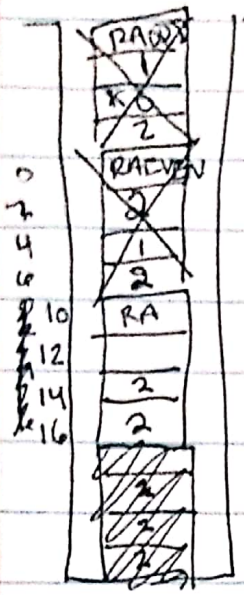
```
Offset
        ---------------------
 0      |  Return address   |   <--- SP
        ---------------------
 2      |  Place for result |
        ---------------------
 4      |  Value of N       |
        ---------------------
 6      |  Value of M       |
        ---------------------
```

8 OLD RA

10 OLD RESULT

12 OLD N

14 OLD M



N=2 R0

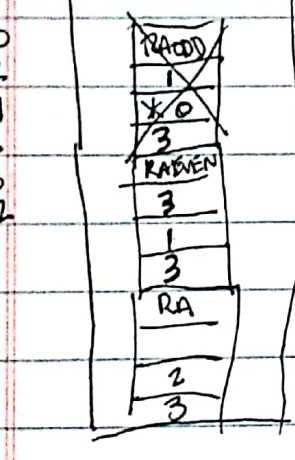
R1

N=2 R2

$2^2 = 4^2$. R3

✓ NZVC

CT



M=3 R0

R1

N=2 R2

$3^2 = 27$? R3

NZVC

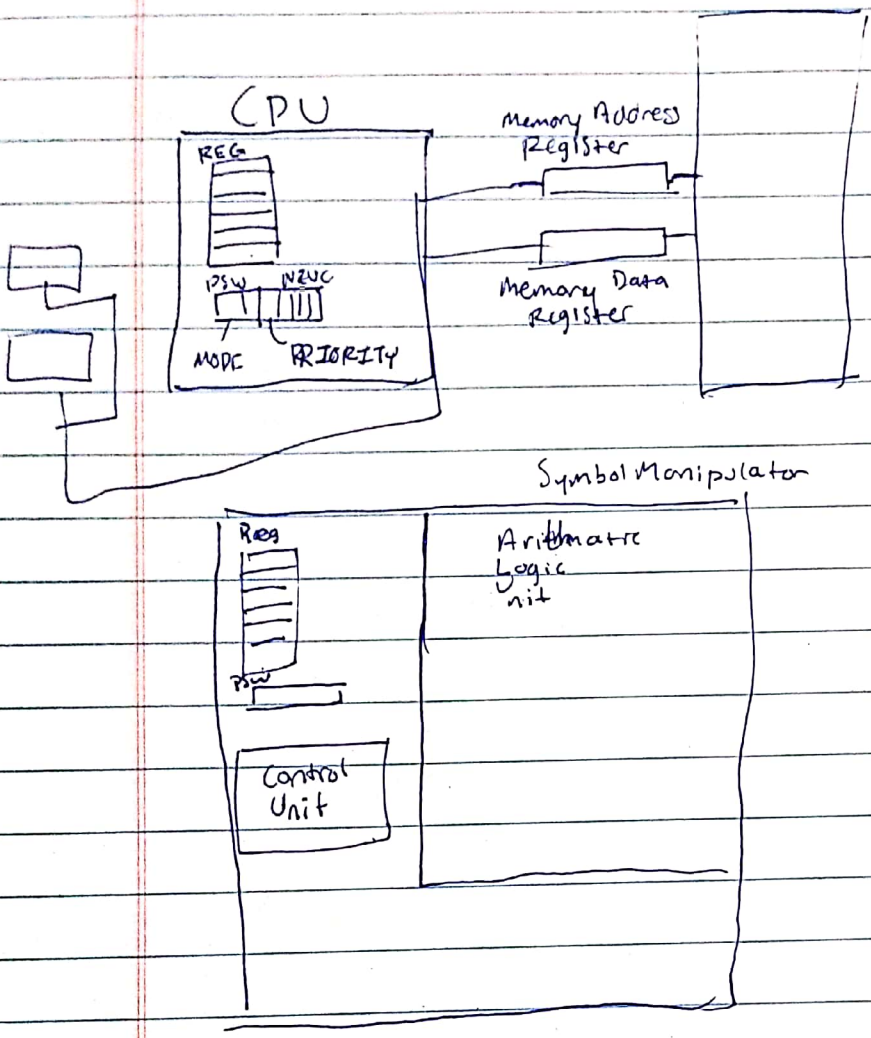
CT

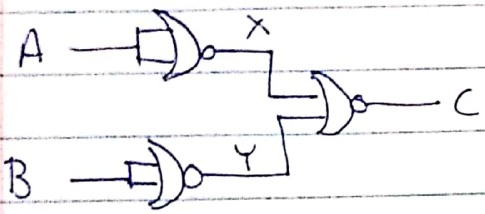
M x N = PROD (Program 1 or something)

Prod: .BLKW 1 ; 16 bit 2's comp non neg
Char: .BLKW 5

express your answer as 5 ASCII characters with leading zeros base 10 and print to screen worth 5 program points

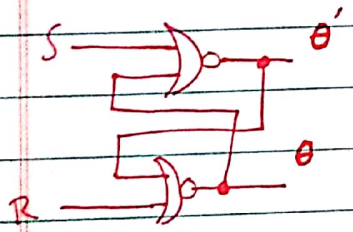
Due Tuesday after break





Using just NOR gates we can build an and gate

A	B	X	Y	C
0	0	1	1	0
0	1	1	0	0
1	0	0	1	0
1	1	0	0	1



S	R	θ	θ'
0	0	0 or 1	(Store)
0	1	0	1
1	0	1	0
1	1		

S M T W Th F S

20 21 22 23

24 25 26 27 28

5

12

19

26

3 4

2 3

Can I get Computer Science research credit for a computer heavy physics research project
- work with physics professor

Can I find project funding as an undergrad?
Get paid for a non credit research project

1. Fetch → inc. PC after moving instruction to instr. reg.
2. ~~Decode~~ decode → is it a good opperand? what is it
3. Execute → Fetch operands, do op., and store result
4. Immediately goto 1.

Clock 

Control Unit → what to do when by manipulating Bus Gates, so the data goes ~~to~~ to the correct register

→ HARDWIRED - built into the hardware with discrete components
- → requires recall for mistakes/design flaws
- → faster than software based Control Unit
- → quickly becomes "too" complicated

→ MICROCODED ("MICROENGINE")
- → program steps rather than hardwired
  - → allows for software patches rather than rewiring
- → CONTROL STORE (OFTEN REWRITABLE)
- → MICROCODE
  - → Player Piano Roll analogy ("Horizontal Microcode")
  - → Defines your instruction set (MUL could be implemented via ~~hardware~~ functional units in the microcode)
  - → Maybe you add MUL FU later, the ~~assembly~~ machine language ~~code~~ doesn't change, the microcode does

The Soul of a New Machine (Computer General / Hardware Bug)

Microcode

"Vertical" → reg. to reg. transfers

→ almost a very very simple machine language

→ more common in modern computers

IBM AS4 compiled to Vertical Microcode

IBM MODEL 360

↳ started the concept of a family of CPUs

↳ all run the exact same machine language

even though the performance is vastly different

PDP-11

80-90 instructions

8 addressing modes

VAX

300+

8

Complex Instruction Set Computer (CISC)

↳ opperations that do much more than one step (eg AVG implemented in microcode)

→ timing differences between operands is much greater

AVG >> DIV >> ASL

↳ AVG >>>> ASL

Pipeline CPU

↳ do more than one operation at a time

F D E

F D E

F D E

Fetch the next instruction when you are decoding the last one.

Decode the next instruction while you execute the next.

3 instructions "running" at the same time.

(Very hard with CISC)

If the adjacent instructions take dramatically different amounts of time, it's very hard to pipeline the CPU

To make pipelining easier, they created RISC

Reduced Instruction Set Computer (ARM) ↳ Instr. Timing very very close

C. mmp - Bill Wulf

The Pentium Chronicles - Bob Colwell (Good Speaker)
↳ Chief Architect of Pentium 4
↳ good book on architecture

RISC (Reduced Instruction Set Computer)
↳ Instruction timing is very close to each other
→ less addressing modes, mostly register to register
↳ very large sets of registers (48, 64 common)
→ can break up into more than 3 phases (FDE)
↳ 4 or 5 so parallization is increased