Non mainstream architectures

Or how politics and economics ruined the computational diversity for the masses

--pancake @ r2con2021

Introduction

Most widely known and used CPU architectures are the Intel and ARM, if you ask around, some people may know mips, ppc or even riscv and avr.

But in the history of computing, many architectures have been abandoned because of technical, political, economical or competitive reasons.

Studying them is interesting and they all shine in some way.

- Dive into the world of abandoned and non-mainstream archs!
- Taking the perspective from r2, and how to support those platforms!

Computing Generations

1st Gen (1940): hermionic vacuum tubes. The IBM 650 lead this era

2nd Gen (1956): Transistors. Much smaller and power efficient. (The IBM 7090)

3rd Gen (1964): Integrated Circuits (silicon) IBM 360/91

4th Gen (1975): VLSI - PC era, very compact, DEC10, PDP11, CRAY-1

5th Gen (1982): ULSI (ultra large scale integration) - AMD/Intel

What's next?

- Quantum? Ternary? Arsenic?

Types of Computers

- General Purpose Computers: Self Contained and focus on user
- SuperComputers: designed for scientific calculations and paralelism, slow data reads, slower writes, but high processing rates.
- Mainframes: designed for stability, used to process large data sets, for statistics, large databases esililenec ad stability, covering data stataisticcs, transactions. IBM leads here.
- HPC - high performance computing - Vector and parallel processing designed for math algorithms. After CRAY, IBM Power9

The 3 racks have a peak performance of 1,48 Petaflops, a 50% more than MareNostrum3 supercomputer, which was uninstalled just one year ago.

The 70's

The glorious days of computing where nothing was written in stone.

In the middle of the cold war, lots of investigations and experiments happening in US and the USSR trying to win the “star wars”.

- Beginning of mainframes
- Not even the binary representation was decided
- Lowercase was not an option, programmers had to scream
- More than two endians!
- Different encodings designed for punchhole hards
- Best Disco hits of all

Setun-70 (aka Сетунь)

During the coldwar both US and USSR developed new technologies and experiments.

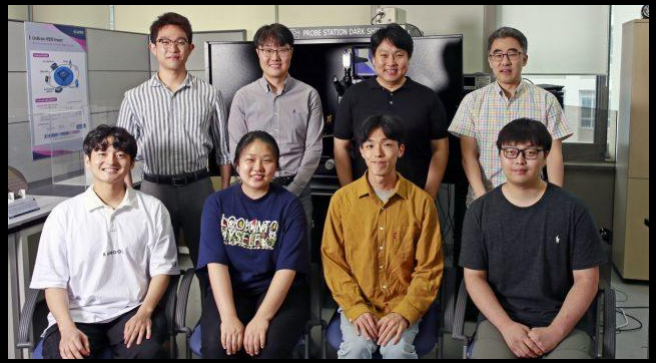
In 1958, Moscow State University build the first ternary computer (team lead by Nikolay Brusentsov).

Lower Power Consumption, Faster math

Using the balanced ternary representation:

- (- 0 +) instead of (0 1 2)

SETUN-70 THE RENAISSANCE



USSR gov forced the ternary computer projects to be abandoned in order to clone what US were doing. (sadness)

But... a couple of years ago: Samsung in 2019 started to develop ternary cpus designed for IoT thanks to its low power consumption:

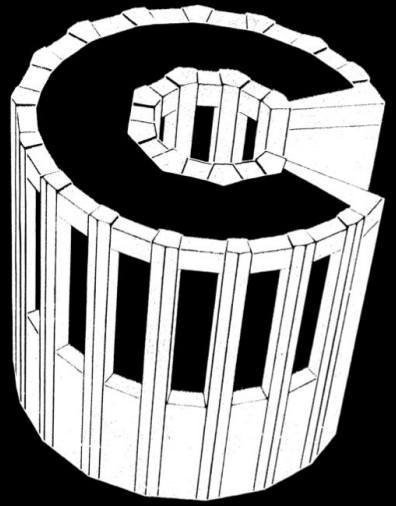
https://www.extremetech.com/computing/295424-back-off-binary-samsung-backed-researchers-debut-ternary-semiconductor

https://hackaday.io/project/164907/logs?sort=oldest

https://www.youtube.com/watch?v=EbJMtJq20NY

```
$ rax2 0t012012
0x8c
$
```

CRAY (1963-1995)



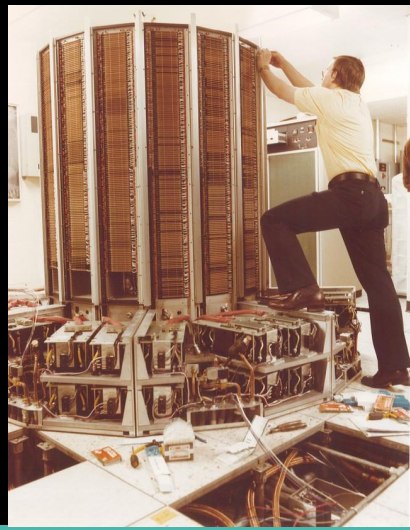
Designed by Seymour Cray in Los Alamos National Lab.

- 64bit cpu at 80mhz with vector processing and 32MB
- Load software from Tapes, not punchcards
- Used VAX or Sun host systems

Runs COS operating system, also UNICOS, based on unix

http://cray.modularcircuits.com

http://www.modularcircuits.com/blog/articles/the-cray-files/downloads/

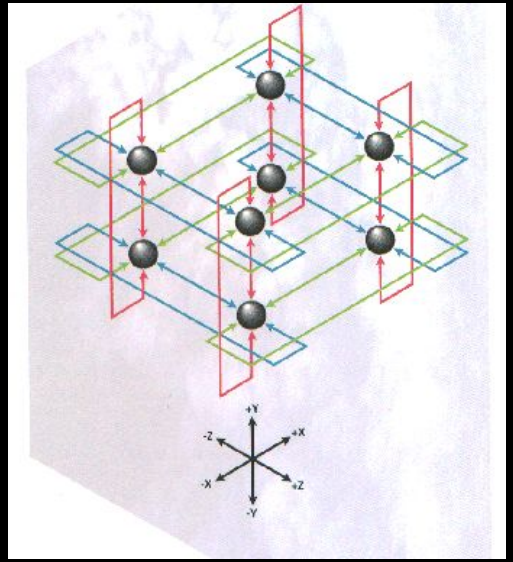


The Fall

CRAY went down just when they were releasing the T3D and were about to start building microprocessors using Gallium Arsenide technologies (instead of silicon semiconductors).

This gas combo performs way better than silicon and produces almost no heat.

This technology seems to get some traction since last year and plans to be widely used in 2026-2030.

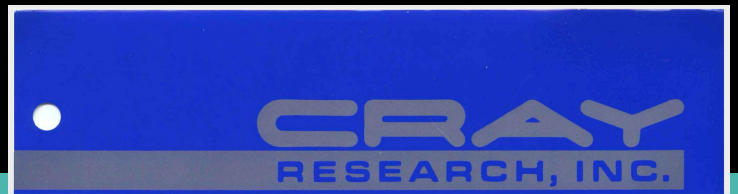


CRAY

Pun intended on the “general-purpose computer”

The CRAY-1 Computer System is a powerful general-purpose computer capable of extremely high processing rates. These rates are achieved by combining scalar and vector capabilities into a single central processor which is joined to a large, fast, bi-polar memory. Vector processing by performing iterative operations on sets of ordered data provide results at rates greatly exceeding result rates of conventional scalar processing. Scalar operations complement the vector capability by providing solutions to problems not readily adapted to vector techniques.

The Cray-3 project—the first attempt at major use of gallium arsenide (GaAs) semiconductors in computing. However, the changing political climate (collapse of the Warsaw Pact and the end of the Cold War) resulted in poor sales prospects. Ultimately, only one Cray-3 was delivered, and a number of follow-on designs were never completed. The company filed for bankruptcy in 1995.



CAS AND UNICOS

C, Fortran, CSH on UNIX for CRAY-1

Disasm looks like asm.pseudo

```
0x0008 (0x000010) 0x2908-       f:00024 d: 264 | OR[264] = A
0x0009 (0x000012) 0x3108-       f:00030 d: 264 | A = (OR[264])
0x000A (0x000014) 0x0809-       f:00004 d:   9 | A = A > 9 (0x0009)
0x000B (0x000016) 0x121F-       f:00011 d:  31 | A = A & 31 (0x001F)
0x000C (0x000018) 0x291D-       f:00024 d: 285 | OR[285] = A
0x000D (0x00001A) 0x700E-       f:00070 d:  14 | P = P + 14 (0x001B)
0x000E (0x00001C) 0x211A-       f:00020 d: 282 | A = OR[282]
0x000F (0x00001E) 0x160F-       f:00013 d:  15 | A = A - 15 (0x000F)
0x0010 (0x000020) 0x82B6-       f:00101 d: 182 | P = P + 182 (0x00C6), C = 1
0x0011 (0x000022) 0x1007-       f:00010 d:   7 | A = 7 (0x0007)
0x0012 (0x000024) 0x291F-       f:00024 d: 287 | OR[287] = A
0x0013 (0x000026) 0x1003-       f:00010 d:   3 | A = 3 (0x0003)
0x0014 (0x000028) 0x2920-       f:00024 d: 288 | OR[288] = A
0x0015 (0x00002A) 0x111F-       f:00010 d: 287 | A = 287 (0x011F)
0x0016 (0x00002C) 0x5800-       f:00054 d:   0 | B = A
0x0017 (0x00002E) 0x1800-0x0F18 f:00014 d:   0 | A = 3864 (0x0F18)
0x0019 (0x000032) 0x7C09-       f:00076 d:   9 | R = OR[9]
0x001A (0x000034) 0x2D1A-       f:00026 d: 282 | OR[282] = OR[282] + 1
0x001B (0x000036) 0x211D-       f:00020 d: 285 | A = OR[285]
0x001C (0x000038) 0x1605-       f:00013 d:   5 | A = A - 5 (0x0005)
0x001D (0x00003A) 0x8402-       f:00102 d:   2 | P = P + 2 (0x001F), A = 0
0x001E (0x00003C) 0x7008-       f:00070 d:   8 | P = P + 8 (0x0026)
0x001F (0x00003E) 0x2118-       f:00020 d: 280 | A = OR[280]
0x0020 (0x000040) 0x1E00-0xFFFF f:00017 d:   0 | A = A - 65535 (0xFFFF)
0x0022 (0x000044) 0x8402-       f:00102 d:   2 | P = P + 2 (0x0024), A = 0
```

```
Unicos/10.0.0: unicos-gen.0 (CRAY J90)

System gen time    = 05/16/98 12:39:45
UNICOS binary size = 2423332 words
Memory Configured  = 4294966272 words
Memory Allowed     = 33552384 words
Memory downed      = 4261412864 words @ address 00177774000
sysmem             = 2185317 words
maxmem             = 28041728 words
Buffer pool size   = 2560000 words (5000 buffers)
User memory avail  = 24006656 words

CPUs configured    = 1, started = 1 (0)

System-Call Timing On

00:00:00(GMT) uts/fs/nc1/nc1vfsops.c-09: INFO Root path is /dev/dsk/root.ram
00:00:00(GMT) uts/fs/nc1/nc1vfsops.c-11: WARNING Cannot open root with disk inc
e.
00:00:00(GMT) uts/fs/nc1/nc1alloc.c-05: WARNING no free space <> on fs <RAMFS>
evice 7
00:00:05(GMT) uts/c1/os/sched.c-13: INFO MPX scheduling is not configurable.
00:00:05(GMT) uts/c1/os/sched.c-11: INFO packtime calc'd as 22 seconds.
00:00:05(GMT) uts/c1/os/sched.c-05: INFO gfactor1_in calc'd as   612 seconds.
00:00:05(GMT) uts/c1/os/sched.c-06: INFO gfactor1_in limited to  410 seconds.
00:00:05(GMT) uts/c1/os/sched.c-10: INFO gfactor1_out set to     0 seconds.
Sat Jan  1 00:00:06 2000
4:@**jklmmlkjjklmmlkjjk4:@++jklmmlkjjklmmlkjjk4:@,,jklmmlkjjklmmlkjjk4:@--jklmm
kjjklmmlkjjk4:@..jklmmlkjjklmmlkjjk4:@//jklmmlkjjklmmlkjjk4:@00jklmmlkjjklmmlkj
k4:@11jklmmlkjjklmmlkjjk4:@22jklmmlkjjklmmlkjjk
INIT: Cannot open /etc/initreq

INIT: failed write of utmp entry: "  "

INIT: failed write of utmp entry: "  "

INIT: SINGLE USER MODE
#
```

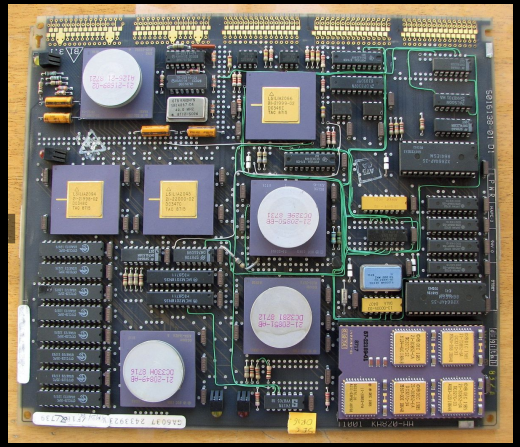
PDP > VAX > ALPHA

Meanwhile in the US, DEC built different RISC computers:

- Pdp11 - lunar lander program (rasm2 -a pdp11 -b 16)
- VAX - 32bit (netbsd runs fine there, therefor r2 too!)
- ALPHA - 1992 - 64bit Windows NT up to 1.1GHz (r2 -a alpha -b64)

VAX

Virtual Address Extension



- 56 byte length instructions
- 32 bit cpu 16 registers
- 4096 byte vector registers
- Orthogonal cisc instruction set
- Several PDP models, from 6-8-12-18 bits
- 1977 - carnegie Melon University
- As fast as s360
- 4 segment registers to address memory (each of 1GB)
- 4 priviledge levels
- Pdp11 compatibility mode

```
1042a:       c0 01           .word 0x01c0 # Entry mask: < r8 r7 r6 >
1042c:       c2 04 5e        subl2 $0x4,sp
1042f:       d0 ac 08 57     movl 0x8(ap),r7
10433:       12 03           bneq 10438 <___start+0xe>
10435:       31 1c 01        brw 10554 <___start+0x12a>
10438:       d0 57 ef c1     movl r7,20800 <__ps_strings>
1043c:       03 01 00
1043f:       d0 a7 08 ef     movl 0x8(r7),20840 <environ>
10443:       f9 03 01 00
10447:       d0 67 50        movl (r7),r0
1044a:       d0 60 51        movl (r0),r1
1044d:       12 03           bneq 10452 <___start+0x28>
1044f:       31 ec 00        brw 1053e <___start+0x114>
10452:       d0 51 ef 77     movl r1,207d0 <__progname>
10456:       03 01 00
10459:       d0 60 51        movl (r0),r1
1045c:       90 61 50        movb (r1),r0
1045f:       13 0f           beql 10470 <___start+0x46>
10461:       d6 51           incl r1
10463:       91 50 2f        cmpb r0,$0x2f
10466:       12 03           bneq 1046b <___start+0x41>
10468:       31 c1 00        brw 1052c <___start+0x102>
1046b:       90 81 50        movb (r1)+,r0
1046e:       12 f3           bneq 10463 <___start+0x39>
10470:       d5 ac 04        tstl 0x4(ap)
10473:       13 0a           beql 1047f <___start+0x55>
10475:       dd ac 04        pushl 0x4(ap)
10478:       fb 01 ff 39     calls $0x1,*207b8 <atexit>
1047c:       03 01 00
1047f:       fb 00 ff 3a     calls $0x0,*207c0 <_libc_init>
10483:       03 01 00
10486:       9e ef 50 02     movab 206dc <__fini_array_end>,r6
1048a:       01 00 56
1048d:       9e ef 49 02     movab 206dc <__fini_array_end>,r0
10491:       01 00 50
10494:       d1 56 50        cmpl r6,r0
10497:       1e 23           bcc 104bc <___start+0x92>
10499:       d0 50 58        movl r0,r8
1049c:       9e ef 3b 02     movab 206dd <__fini_array_end+0x1>,r0
104a0:       01 00 50
104a3:       c2 50 58        subl2 r0,r8
104a6:       ca 03 58        bicl2 $0x3,r8
104a9:       9e 48 ef 30     movab 206e0 <__CTOR_LIST_END__>[r8],r8
104ad:       02 01 00 58
104b1:       d0 86 50        movl (r6)+,r0
104b4:       fb 00 60        calls $0x0,(r0)
104b7:       d1 56 58        cmpl r6,r8
104ba:       12 f5           bneq 104b1 <___start+0x87>
104bc:       9f cf 2e ff     pushab 103ee <___start+0xe>
104c0:       fb 01 ff f1     calls $0x1,*207b8 <atexit>
104c4:       02 01 00
104c7:       9e ef 0f 02     movab 206dc <__fini_array_end>,r6
104cb:       01 00 56
104ce:       9e ef 08 02     movab 206dc <__fini_array_end>,r0
104d2:       01 00 50
104d5:       d1 56 50        cmpl r6,r0
104d8:       1e 23           bcc 104fd <___start+0xd3>
104da:       d0 50 58        movl r0,r8
104dd:       9e ef fa 01     movab 206dd <__fini_array_end+0x1>,r0
104e1:       01 00 50
104e4:       c2 50 58        subl2 r0,r8
104e7:       ca 03 58        bicl2 $0x3,r8
104ea:       9e 48 ef ef     movab 206e0 <__CTOR_LIST_END__>[r8],r8
104ee:       01 01 00 58
104f2:       d0 86 50        movl (r6)+,r0
104f5:       fb 00 60        calls $0x0,(r0)
104f8:       d1 58 56        cmpl r8,r6
```

r2@netbsd/vax

$ brew install simh

Very portable and fast emulator for all the PDP and VAX machines from DEC, developed by Robert M. Supnik.

- Data General Nova, Eclipse
- Digital Equipment Corporation PDP-1, PDP-4, PDP-7, PDP-8, PDP-9, PDP-10, PDP-11, PDP-15 (and UC15), VAX11/780, VAX3900
- GRI Corporation GRI-909, GRI-99
- IBM 1401, 1620, 7090/7094, System 3
- Interdata (Perkin-Elmer) 16b and 32b systems
- Hewlett-Packard 2114, 2115, 2116, 2100, 21MX, 1000, 3000
- Honeywell H316/H516
- MITS Altair 8800, 8080 only
- Royal-Mcbee LGP-30, LGP-21
- Scientific Data Systems SDS 940
- Xerox Data Systems Sigma 32b systems

```
$ cat netbsd-boot.vax
load -r ka655x.bin
set cpu 64m
set rq0 ra92
at rq0 netbsd.dsk
set rq1 cdrom
at rq1 NetBSD-9.2-vax.iso
at xq0 en0
boot cpu
boot dua0:
# boot from DISK
# boot from CD
# boot dua1:
$ cat Makefile
all:
        vax netbsd-boot.vax
$
```

MIDDLE ENDIAN

The legend say there are more than two endians.. Well, i'm sorry to say that unfortunately the only middle endian cpu was the PDP.

- PDP-11 operates instructions in many endians, mainly little
  - Some extended FPU instructions use middle-endian encoding
  - Those 1234 4321

Printing endian unicorns with r2!

```
$ r2 /bin/ls
[0x100001060]> pve 4321
0x100001060  -443987883 (0xe5894855)
[0x100001060]> pve 1234
0x100001060  1430817253 (0x554889e5)
[0x100001060]> pve 1324
0x100001060  1435060453 (0x558948e5)
[0x100001060]>
```

Mainframes

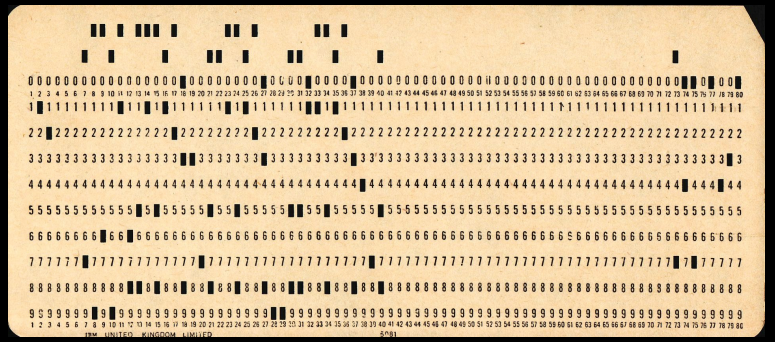
MVS

MVS is a very old operating system, and it is an evolution of older operating systems (!!). Its father is OS/360 and its grandfather is DOS/360. Note that the Apollo mission was conducted by a S/360 mainframe, using some sort of variation or similar mutation of them.

It has evolved during time and it is completely retro-compatible with your favorite punch-cards drilled by your great grandfather with a piece of bone. And now:

- Have you noticed that the default text screen size in the IBM/PC is 80x25?
- Have you noticed that the default text width in a 3270 terminal is always 80 characters?
- Have you noticed that an IBM punch card has exactly 80 columns.

This is Sparta.



MVS

In the early 1970s IBM introduced virtual memory (then called "virtual storage”), and it makes possible to

- allocate space larger than available RAM,
- reduce impact of memory fragmentation, and
- improve security.

IBM run into the absolute overwhelming naming madness of the infinite and on each new hype-feature it created more and more operating systems. Fetch’em all:

OS/VS2R2 → MVS/SE → MVS/SE2 → MVS/SP → MVS/XA → MVS/ESA → OS/390 → z/OS

And in parallel an innumerable list of utilities and facilities with unintelligible acronyms were created to increase the confusion level to an unexpected level never seen before in the world of computer science and never seen again until the launch of SAP products:

> ACF/TCAM, ACF/VTAM, DF/DS, DF/EF, DF/DSS DFS, SAM-E, MVS/370 DFP, MVS/XA DFP Version 1 Release 1, MVS/XA DFP Version 2 Release 1, MVS/ESA DFP Version 3, DFSMS, OS/VS2 MVS TSO, TSO Command Processor, TSO/VS2 Programming Control Facility, TSO PCF II, TSO Extensions, OS/VS2 MVS/System Extensions, MVS/System Product (introducing his friends JES3 Version 1, JES2 Version 1, MVS/System Product-JES2 Version 2, MVS/System Product-JES3 Version 2, MVS/System Product-JES2 Version 3, MVS/System Product-JES3 Version 3, MVS/ESA System Product: JES2 Version 4, MVS/ESA System Product: JES3 Version 4, MVS/ESA System Product: JES2 Version 5 and MVS/ESA System Product: JES3 Version 5, Interactive System Productivity Facility (ISPF), DASD, DFSMS, RACF, APL, CICS, CICS DDM, MQ, REXX, z/VM, z/VSE and more

MVS

Retrocompatibility and not rewriting critical business applications written in COBOL on punch cards is a great reason to not rewrite from scratch apps and moving to newer systems.

But it is not the only reason to ride on a dinosaur:

- Address space was increased to 24 bit (MVS), 31 bit (s/390) and 64 bit (z/OS)
- Preemptive execution and memory protection.
- Virtualization (Would you like to hear about the z/VM family and the thousand acronyms it received before?)
- LPAR mainframe partitioning
- Batch processing and workload management
- Mainframe Sysplex
- A richful, overwhelming hyper interactive and extremely funny rich text-based user interfaces (3270 and EBCDIC).
- Transactional management systems (CICS, IMS/TM)
- Rock-solid databases (DB2, IMS/DB)
- At your option an extremely and simple stupid permission system (the default VMS model), or a high-doped-on-steroids security subsystem (RACF, CA/TopSecret)
- Unix System Services (a z/OS process running and entire Unix subsystem)

HOW TO FIT r2 IN MVS?

Building r2 for zOS is not possible because its not a posix system

- We can build and run small programs in C
- Programs are not interactive, but could use ??? Apis for tui
- Execution happens as jobs
- What about a rap server? We need tcp/ip or serial..
- Memory layout? Check self:// for inspiration
- What about porting r0? We still need an s370-390 tiny disassembler

REXX on MVS

Its an ancient language that runs anywhere and have been implemented several times for many operating systems

Based on the gnu regina rexx implementation i wrote the r2pipe api

- r2pipe.rexx

```
Say r2cmd('?E hello from rexx')

Exit

r2cmd: PROCEDURE EXPOSE globals.
  arg cmd
  fin = '/dev/fd/'Getenv(R2PIPE_IN)
  fou = '/dev/fd/'Getenv(R2PIPE_OUT)
  o = CharOut(fou, cmd''D2C(0))
  len = 0
  DO while len == 0
    len = Chars(fin)
  END
  return CharIn(fin,,len)
```

ZARCH/S360/S370/S390

Depending on the cpu model you'll need to use a different configuration for the disassembler.

- R2 -a s390 -e asm.cpu=?
  - Previously named zarch from capstone
- R2 -a s390.gnu -e asm.cpu=?

```
[0x000005f0]> e asm.cpu=?
esa
zarch
[0x000005f0]> e asm.cpu=zarch
[0x000005f0]> pd 10
            ;-- entry0, section..text, sym..text, sym._start:
            0x000005f0      la    %r4,8(%r15)
            0x000005f4      invalid
            0x000005f5      unaligned
            0x000005f6      invalid
            0x000005f7      unaligned
            0x000005f8      invalid
            0x000005f9      unaligned
            0x000005fa      lghi  %r0,-16
            0x000005fe      ngr   %r15,%r0
            0x00000602      aghi  %r15,-176
[0x000005f0]> e asm.cpu=esa
[0x000005f0]> pd 10
            ;-- entry0, section..text, sym..text, sym._start:
            0x000005f0      la    %r4,8(%r15)
            0x000005f4      invalid
            0x000005f5      unaligned
            0x000005f6      invalid
            0x000005f7      unaligned
            0x000005f8      invalid
            0x000005f9      unaligned
            0x000005fa      invalid
            0x000005fb      unaligned
            0x000005fc      invalid
[0x000005f0]> e asm.arch=s390
[0x000005f0]> pd 10
            ;-- entry0, section..text, sym..text, sym._start:
            0x000005f0      la    %r4,  8(%r15)
            0x000005f4      lg    %r3,  0(%r15)
            0x000005fa      lghi  %r0,  -0x10
            0x000005fe      ngr   %r15, %r0
            0x00000602      aghi  %r15, -0xb0
            0x00000606      xc    0(8, %r15), 0(%r15)
            0x0000060c      stmg  %r14, %r15, 0xa0(%r15)
            0x00000612      la    %r7,  0xa0(%r15)
            0x00000616      larl  %r6,  sym.__libc_csu_fini
            0x0000061c      larl  %r5,  sym.__libc_csu_init
[0x000005f0]>
```

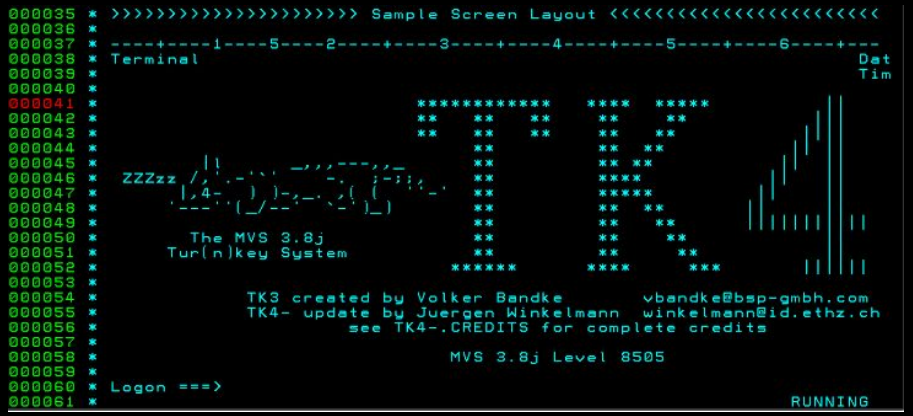
MVS HELLO WORLD

```
000014  0072  ****                                  AL2(114),C'...'    Saved Options String
                Timestamp and Version End

1                                                   ./prueba.c: main                04/30/21 09:4
 5:24   Page     5

OFFSET OBJECT CODE        LINE#  FILE#    P S E U D O   A S S E M B L Y   L I S T I N G

                          000001 |       *  #include <stdio.h>
                          000002 |       *
                          000003 |       *  int main()
000088                    000003 |       main      DS      0D
000088  47F0  F024        000003 |                 B       36(,r15)
00008C  0C3C5C5C                                  CEE eyecatcher
000090  00000A0                                   DSA size
000094  00000090                                  =A(PPA1-main)
000098  47F0  F001        000003 |                 B       1(,r15)
00009C  58F0  C31C        000003 |                 L       r15,796(,r12)
0000A0  184E              000003 |                 LR      r4,r14
0000A2  05EF              000003 |                 BALR    r14,r15
0000A4  00000000                                  =F'0'
0000A8  A7F4  000C        000003 |                 J       *+24
0000AC  90ES  D00C        000003 |                 STM     r14,r5,12(r13)
0000B0  58E0  D04C        000003 |                 L       r14,76(,r13)
0000B4  4100  E0A0        000003 |                 LA      r0,160(,r14)
0000B8  5500  C314        000003 |                 CL      r0,788(,r12)
0000BC  A724  FFF0        000003 |                 JH      *-32
0000C0  58F0  C280        000003 |                 L       r15,640(,r12)
0000C4  90F0  E048        000003 |                 STM     r15,r0,72(r14)
0000C8  9210  E000        000003 |                 MVI     0(r14),16
0000CC  50D0  E004        000003 |                 ST      r13,4(,r14)
0000D0  18DE              000003 |                 LR      r13,r14
0000D2  C030  0000  0019  000003 |                 LARL    r3,F'25'
0000D8                    End of Prolog

0000D8  C050  0000  0018  000000 |                 LARL    r5,F'24'
                          000004 |       *  {
                          000005 |       *     printf("Hola mundo\n");
0000DE  58F0  3000        000005 |                 L       r15,=V(printf)(,r3,0)
0000E2  1805              000005 |                 LR      r0,r5
0000E4  4110  D098        000005 |                 LA      r1,#MX_TEMP1(,r13,152)
0000E8  5000  D098        000005 |                 ST      r0,#MX_TEMP1(,r13,152)
0000EC  0DEF              000005 |                 BASR    r14,r15
                          000006 |       *     return(0);
0000EE  41F0  0000        000006 |                 LA      r15,0
                          000007 |       *  }
0000F2                    000007 |       @1L1     DS      0H

0000F2                    Start of Epilog
0000F2  180D              000007 |                 LR      r0,r13
0000F4  58D0  D004        000007 |                 L       r13,4(,r13)
0000F8  58E0  D00C        000007 |                 L       r14,12(,r13)
0000FC  9825  D01C        000007 |                 LM      r2,r5,28(r13)
000100  051E              000007 |                 BALR    r1,r14
000102  0707              000007 |                 NOPR    7

000104                    Start of Literals
000104  00000000                                  =V(printf)
000108                    End of Literals

                          ***   General purpose registers used: 1101110000001111
                          ***   Floating point  registers used: 1111111100000000
                          ***   Size of register spill area: 128(max) 0(used)
                          ***   Size of dynamic storage: 160

1                                                   ./prueba.c: main                04/30/21 09:4
 5:24   Page     6

OFFSET OBJECT CODE        LINE#  FILE#    P S E U D O   A S S E M B L Y   L I S T I N G

                          ***   Size of executable code: 124
```

```
[0x00001928 [xAdvc]0 0% 310 prueba]> pd $r @ hit0_0
            ;-- hit0_0:
            0x00001928          b       0x24(%r15)
            0x0000192c          invalid
            0x0000192d          unaligned
            0x0000192e          bprp    0xc,  0x232e, 0x1a6e
            0x00001934          invalid
            0x00001935          unaligned
            0x00001936          invalid
            0x00001937          unaligned
            ;-- hit0_1:
            0x00001938          b       1(%r15)
        --> 0x0000193c          l       %r15, 0x31c(%r12)
            0x00001940          lr      %r4,  %r14
            0x00001942          balr    %r14, %r15
            0x00001944          invalid
            0x00001945          unaligned
            0x00001946          invalid
            0x00001947          unaligned
        ,=< 0x00001948          j       0x1960
        |   0x0000194c          stm     %r14, %r5,  0xc(%r13)
        |   0x00001950          l       %r14, 0x4c(%r13)
        |   0x00001954          la      %r0,  0xa0(%r14)
        |   0x00001958          cl      %r0,  0x314(%r12)
        |=< 0x0000195c          jh      0x193c
        `-> 0x00001960          l       %r15, 0x280(%r12)
            0x00001964          stm     %r15, %r0,  0x48(%r14)
            0x00001968          mvi     0(%r14), 0x10
            0x0000196c          st      %r13, 4(%r14)
            0x00001970          lr      %r13, %r14
            0x00001972          larl    %r3,  0x19a4
            0x00001978          larl    %r5,  0x19a8
            0x0000197e          l       %r15, 0(%r3)
            0x00001982          lr      %r0,  %r5
            0x00001984          la      %r1,  0x98(%r13)
            0x00001988          st      %r0,  0x98(%r13)
        ,=< 0x0000198c          basr    %r14, %r15
        |   0x0000198e          la      %r15, 0
        |   0x00001992          lr      %r0,  %r13
        |   0x00001994          l       %r13, 4(%r13)
        |   0x00001998          l       %r14, 0xc(%r13)
        |   0x0000199c          lm      %r2,  %r5,  0x1c(%r13)
        |   0x000019a0          balr    %r1,  %r14
        |   0x000019a2          bcr     0,    %r7
        |   0x000019a4          invalid
        |   0x000019a5          unaligned
        |   0x000019a6          invalid
        |   0x000019a7          unaligned
        |   0x000019a8          invalid
        |   0x000019a9          unaligned
        |   0x000019aa          invalid
        |   0x000019ab          unaligned
        |   0x000019ac          sth     %r9,  0x495(%r4, %r10)
        |   0x000019b0          brxh    %r9,  %r6,  0x43b0
        |   0x000019b4          invalid
        |   0x000019b5          unaligned
        |   0x000019b6          invalid
        |   0x000019b7          unaligned
        |   0x000019b8          mr      %r12, %r14
```

TK-4 - Tur(n)key-4 system

- MVS 3.8j distro maintained by the community
- Very outdated MVS, but at least public domain, thanks to the US gov who payed to IBM to get those technologies moving forward and more accessible for students.
- Images can corrupt if shutted down incorrectly
- Can run virtualized linux systems
- Setting up TCP/IP is very hard
- Passwords are case insensitive
- Fortran, REXX, Assembly and C



CHARSETS: EBCDIC37

EBCDIC37 is optimized for punchcards, to reduce the possibility to break the paper (...) may be better for memory too?

- xxd -E (standard flag)
- Using r2 strings (see tests)
- e cfg.charset=ebcdic37

```
[0x00000000]> s 0x19a8
[0x000019a8]> pd 2
            0x000019a8      enter 0xffffffffffff9396, 0xffffffffffffff81
            0x000019ac      xchg  esp,  eax
[0x000019a8]> x
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x000019a8  c896 9381 4094 a495 8496 1500 0000 0000  ....@...........
0x000019b8  1cce a106 0000 00d0 0000 0000 0000 0000  ................
0x000019c8  ff00 0000 0000 0000 9000 0000 0040 0012  .............@..
0x000019d8  0000 0000 5000 003e ffff ff70 3828 0000  ....P..>...p8(..
0x000019e8  4009 0035 0000 0000 0004 9481 8995 0000  @..5............
0x000019f8  0300 2203 ffff fea8 0000 0000 ffff fea8  ..".............
0x00001a08  0000 0000 0200 0000 0100 0001 0000 0088  ................
0x00001a18  0000 0090 0000 0000 d6f0 f0f3 0000 00e4  ................
0x00001a28  0003 0000 0000 0000 0000 0000 0000 0000  ................
0x00001a38  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x00001a48  0000 0000 0000 0000 0000 13a8 0000 12d0  ................
0x00001a58  0000 11f8 0000 0000 0000 0008 0000 0010  ................
0x00001a68  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x00001a78  0000 1120 0000 0018 0000 1048 0000 0020  ... .......H... 
0x00001a88  0000 0f70 0000 0028 0000 0e98 0000 0030  ...p...(.......0
0x00001a98  0000 0dc0 0000 0038 0000 0ce8 0000 0040  .......8.......@
[0x000019a8]> e cfg.charset =?
ascii
ebcdic37
pokered
[0x000019a8]> e cfg.charset =ebcdic37
[0x000019a8]> ps
Hola.mundo
<null><null><null><null><null>????<null><null><null>?<null><null><null><
ll><null><null><null><null><null><null>?<null><null><null><null>.<null>?
ull>???????<null><null>.?<null>?<nu
[0x000019a8]>
```

```
NAME=ebcdic37 hexstr
FILE=bins/s390/zos/prueba/pi
ARGS=-n
CMDS=<<EOF
s 0x19a8
px 16
ps
e cfg.charset=ebcdic37
px 16
ps 10
EOF
EXPECT=<<EOF
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x000019a8  c896 9381 4094 a495 8496 1500 0000 0000  ....@...........
\xc8\x96\x93\x81@\x94\xa4\x95\x84\x96\x15
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x000019a8  c896 9381 4094 a495 8496 1500 0000 0000  Hola_mundo.?????
Hola_mundo
EOF
RUN
```

To Learn more on mainframes

- Watch the videos from Moshix
- Join the Discord server
  - https://discord.gg/DxWdQMFT
- Install TK-4 (TurnKey4 System)
  - MVS 3.8j distribution maintained by mainframe fans
- Use the Hercules.. Well Hyperion emulator to run it

other archs

Amiga

Very successful and flexible OS and hardware, based on m68k, but later on PowerPC (WarpOS).

- Supported parallel boards running side-by-side with workbench
- Redesign when the dog sits, each chip does a specific task
- R2 supports m68k as well as basic support for HUNK executable files

Amiga community is still very active and passionate, so supporting r2 in there is more

Fantasy computers

PICO8, B8, Basic8, Lambda8, Lico12, Retro40 and TIC-80.

I focused on the last one because is the most free-software friendly.

- 8/16bit hardware specs
- VM runs Gzip (Lua, JS, Moonscript, ..)
- TIC format supported by r2

IO Banks

An IOBank in r2 is a new api and commands that group maps with a specific priority order. Those can be used in many ways:

- Gameboy memory bank
- R0-3 context switching for kernel emulation
- Segment prefixed memory access (thread local storage)

Use the ‘omb’ command to manipulate them.

V850

Widely used in the automotive industry and other realtime critical systems.

- Supports large ranges of temperatures
- Parallel/sync execution
- Cyclic memory, for fun optimizations with negative references
- 26bit memory references, using 2 lower bits to define cpumode
  - https://mobile.twitter.com/trufae/status/1367541260236566538
- Switch instruction in E2 models
- Jump table analysis with esil
- Disassembler in r2 supports first vanilla generations
- R2ghidra support for disasm, analysis, emulation and decompilation

Finding UDS CAN tables with /ru

cyclic memory

The v850 cpu implements a cyclic memory model.

This means that the compiler can do optimizations like:

- use a negative relative reference from lower addresses near 0 to reference the memory mapped devices.
- This saves some precious bytes in the final assembly

Only the first block allows code execution, but data and devices can be used at any of the cycles.

This is supported in r2 with `e io.mask=0x3fffffff`

M32C

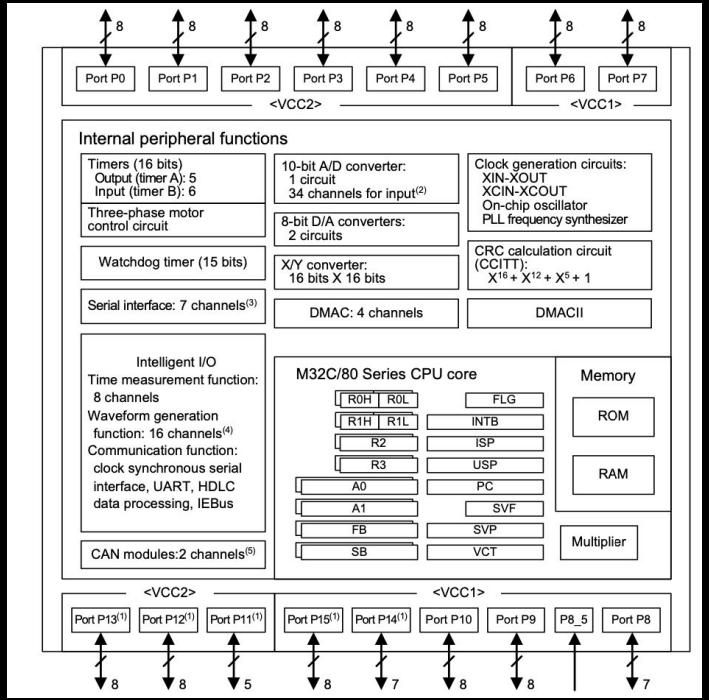
32bit microcomputer from Renesas

- 16MB address space
- For high-speed processing applications
  - Office automation, comm and industrial devices
  - Widely used in medical devices

Only supported in IDA

- No sleigh or other tool
- Just rip the code from binutils.. Oh wait!

https://sourceware.org/cgen/



Register windows

SPARC processors use register windows. Each register window consists of eight **in** registers, eight **local** registers, eight **out** registers, and eight **global** registers. Out registers are the in registers for the next window. The number of register windows ranges from 2 to 32, depending on the processor implementation. (up to 520 registers, commonly being able to handle 8 windows)

- CWP register moves the window
- Each window handles 8 input, 8 output and 8 global registers to use in fcn
- Feature not supported in RReg, anyone? :D

Thanks for watching!

1960

- risc cpu from intel 10-100mhz
-

File Formats