**Round-about Architecture**

*Walter B. Gress V*

*Wednesday, March 27, 2018*

An improvcement on the modern day CPU pipeline using the ceoncept of traffic round-abouts and an extension of the instruction set

to object oriented, etc. high level instructions.

# introduction

When you have a bottleneck what do you do about traffic? You could add a traffic light, add a secondary road, increase or decrease the speed limit, or perhaps widen the road. All have been solutions tested with the Von Neumann bottleneck. However, what if the bottleneck was a round-a-bout?

# ARCHITECTURE

The Von Neumann architecture consists of multiple address busses and a control unit, arithmetic logic unit, memory unit, and input and ouput units.We suggest a circular round-a-bout architecture. The address bus consists of registers in a circular fashion through which packets of information are sent from register to register in a ring like pattern. Adjacent to the bus are your standard input registers, output registers, CU (control unit), ALU (atrithmetic logic unit), and memory.

See Figure 3.

# packets

Each packet has the following structure:





Figure 1

%c0

mov 5 %a1

Figure 2

A is the target register for the packet to move to. B is the opcode, with three optional parameters. C is the current register the packet is residing on.

The registers that compose the ring are labelled %a0 through a%10 (or more). Each packet moves from register to register until it gets to the target register, that is, the register adjacent to the unit (for example, moving from %a0 to % 5 to reach control unit %c0).

Registers can contain both data (like the packet above) or commands. If a packet is moved into an external register (one not on the bus), for example %c0, it will perform the operation.

The registers for the architecture are as follows:

%a address bus

%c control unit

%u artithmetic logic unit

%n memory

%o output

%i input

%s string operations and shifts



%sp stack pointer (points to top of stack)

%bp points to base of stack frame

%si points to source in stream operations

%di points to destination in stream operations

%ip instruction pointer (current instruction)

%cc stores the set condition code

%fl flags status register

All registers are 64 bit. Pointers are 8 bytes wide. Pushes and pops to the stack are in 8 byte strides.

Copy opcode example:

%c0 | cop 5 %o 0x100 %a0

Where %c0 is the control unit register, 5 is a constant, %o is the output register, 0x100 is the location on the output bank to copy to and %a0 is the current register on the address bus the packet is sitting on.

This packet is being routed to the control unit because the control unit con trolls the opcode cop. Another example is:

%u0 | add 5 10 %n1 0xFFF %a1

This packet is sitting on the address bus %a1, is going to the ALU for add operation and is adding 4 to 10 and putting the result in the bank 0xFFF via %n1 memory register.

FLAGS

CF Carry Flag 0x0001

PF Parity Flag 0x0004

ZF Zero Flag 0x0040

SF Sign Flag 0x0080

TF Trap Flag 0x0100

IF Interrupt Enable Flag 0x0200

DF Direction Flag 0x0400

OF Overflow Flag 0x0800

IOPL No Privilege Level 0x3000

NTA Nested Task Flag 0x4000

EFLAGS

RF Resume Flag 0x001 000

VM Virtual Mode 0x003 000

AC Alignment Check 0x0004 000

VIF Virtual Interrupt Flag 0x0008 000

VIP Virtual Interrupt Pending 0x0010 000

ID Able to use CPID Instruction 0x0020 000

**OPCODES**

**LOGICAL (ALU)**

and logical and

or logical or

nand logical not a nd

nor logical not or

not logical not

xor logical xor

nxor logical nxor

andcc logical and set condition code

orcc logical or set condition code

nandcc logical not and set condition code

norcc logical not or set condition code

notcc logical not set condition code

xorcc logical xor set condition code

band bitwise and

bor bitwise or

bxor bitwise xor

nbnd bitwise nand

nbor bitwise nor

nxor bitwise nxor

mask apply a mask to a register

**ARITHMETIC (ALU)**

add add

sub subtract

subu subtract without borrow

div divide

mult multiply

umult unsigned multiply

udiv unsigned divide

exp exponentiate

mod modulo operator

sumt summation

dec decrement

inc increment\

fabs absolute value

csgn change sign

divp divide and pop

dvp2 divide and pop twice

mltp multiply and pop

mlt2 multiply and pop twice

**FLOATING POINT UNIT**

fadd floating point add

fsub floating point subtract

fsubu floating point subtract without borrow

fdiv floating point division

fmult floating point multiplication

exp floating point exponentiation

sumtf floating point summation

divr divide reversed

dvrp divide reversed and pop

mltr multiply reversed

mtrp multiply reversed and pop

lg2e load log\_2(e) on stack

lg210 load log\_2(10) on stack

lg102 load log\_10(2) on stack

ln load ln 2 on stack

lpi load PI onto stack

loe load e onto stack

sin perform sin operation

cos perform cos operation

tan peform tan operation

sinh perform hyperbolic sin

cosh perform hyperbolic cos

tanh perform hyperbolic tan

cosin peform inverse cosine

insin perform inverse sin

intan perform inverse tan

trunc truncate

tayl peform taylor series

sqrt perform square root

**BRANCH (CONTROL UNIT)**

bl branch on less

ble branch on less than or equal

be branch of equal

ba branch always

bn branch never

bne branch on not equal

bge branch on greater than or equal

bg branch on greater than

bt branch on true

bf branch on false

int call to interrupt

ret return from procedure

wait wait until not busy

**SHIFT (ALU)**

sll shift left logical

srl shift right logical

sra shift right arithmetic

sla shift left arithmetic

clr clear a register to 0

call calls a function

rl rotate left with carry

rr rotate right with carry

**MISC (CONTROL UNIT)**

nop no operation

cmp subtract from register, put result in register

lbl a label for a branch to jump to

cbw convert byte to word

clfl clear a flag

cupid returns processor specific info

rtime returns time since activated

**CONDITION CODE (CONTROL UNIT)**bneg branch on negative

bpos branch on positive

bz branch on equal to zero

bnz branch on not equal to zero

bvs branch on overflow set

bvc branch on overflow clear

bcs branch C set

bcc branch C clear

**MEMORY (CONTROL UNIT)**

ld load a value to a register

save save a value to a register

mov move from register to register

pop pop data from stack

push push data onto stack

popa pop all general purpose registers from stack

bswap byte swap

wbic write back and invalidate cache

invd flush internal cache

mov move from register to register

xchg exchange values in registers

cmvl conditional move less than

cmvle conditional move less equal

cmve conditional move equal

cmvge conditional move greater equal

cmvg conditional move greater

mskm masked move

cmpp compare and pop

cmp2 compare and pop twice

**WORD (CONTROL UNIT)**

cbtw convert byte to word

cwtd convert word to double word

cdwq convert double word to quad

**CRYPTOGRAPHY (FPU)**

aesync perform one round of enc AES flowae

asyncl perform last round of encAES flow

aedync perform one round of dec AES flow

dsyncl perform last round of dec AES flow

aesrn assist in AES round key generation

aemix assist in AES inverse mix columns

**GRAPHICS (GPU)**

stmvm store model view matrix

proj projection matrix

mvproj model view projection matrix

local store local coordinates

polyst store polygon

trist store triangle

stindx store index for indexed triangles

norml calculate normal for triangle

distv distance vector

exp2 exponential base 2

floor floor function

fract store number as a fraction

lit compute light coeffcients

mad multiply and add

max maximum

min minimum

rec recipricol

recsq recipricol square root

textr texture lookup

texst texture store

linint linear interpolation

ceil ceiling

scmult scalar multiplication to matrix

modsc modulous vector components by scalar

refv reflection vector

round round to nearest integer

sad sum of absolute differences

sign set sign

samp texture sample

txf texel fetch

texld texture with level of detail

mips multi-texture with distance

diff diffuse value front

spec specular value front

amb ambient value front

emiss emission front

shini shininess front

diff diffuse value back

spec specular value back

amb ambient value back

emiss emission back

shini shininess back

arrpoly create a poly coord buffer

clip perform clipping plane operations

frust set view frustrum

fogc fog color

fogi fog intensity

pnt point x,y,z store in register

sptlt spot light

ptlt point light

amblt ambient light

**EXTENDED MATH OPCODES (FPU)**

dotpr multiply two vectors (dot product)

cross multiply two vectors (cross product)

deter calculate the determinant

stm store matrix

mltm matrix multiplication

egve calculate eigenvector

egva calculate eigenvalue

**OBJECT ORIENTED OPCODES**

**OO UNIT**

beint begin interface code

eeint end interface code

bclc begin class code

eclc end class code

inhc inherit from class at address

inhi inherit from interface at address

virt virtual function block

ovr override function block

for for block

forech foreach block

nmspc namespace block

while while loop block

static static block

obj object instance

ovr overloaded operator block

thrd thread block

meta metadata block

prpty property block

tmpt template block

enum stores an enumeration

list stores a list of elements

hahsm stores a hashmap of elements

arry stores an array

exch for exception handling

embd embedded (anonymous) methods

**STRINGS (ALU)**

cpystr copies a string from reg to reg

mvstr moves a string from reg to reg

strl returns string length

token string tokenizes into memory

ldstr load string byte

ldstrw load string word

ldstdw load string doubjle word

ldstrd load string quadword

bstre branch string equal

bstrne branch string not equal

# PRIORITY QUEUE OF STACKS

Each queue is a thread. Each hashtable is a hashtable with buckets.

One stack for function DrawLine():











Queue of stacks. Instead of a single pile of stacks, we use a threaded priority queue of stacks.







**A**



Output %o

Graphics Unit %g

Memory %n

Control unit %c

Outer Loop

ALU %u

FPU %f

a, b, c. Mutex is locked on b by B

B

a,b,c. Mutex is locked for a by A

a,b,c. Mutex lock using b.

# A more advanced architecture

ROUND-A-BOUT CORE

OO Unit %z

Address Bus

%a

Input %i

ed.

The inner loop is added so that if there is congestion on the outer loop, the packet can switch lanes so to speak. The additional processing units have also been add

Synchronization

Unit

Main Address Bus

L1 Cache

L2 Cache

# PSEUDO Fetch-DECODE-Execute Cycle

1. Rotate – The values on the address bus are updated. For example %r0 is moved to %r1, %r1 to %r2, etc. If there is a roadblock, the packet is moved to the inner round-a-bout. If the instruction is on the inner round about and the outer is open it is moved to the outer roundabout. If a packet has reached its target register on a unit, the packet is moved to that register.