**Axomatic Processor Architecture**

**Jim Brakefield Mar 2 11:50AM**

Partially as a response to Quadibloc & Skybuck Flying and partially as a result of my own interests the following is submitted for your review and comment.   
  
I would like nothing better than to be on an ad-hoc committee that meets in various exotic locations and argues the details.   
  
Introduction   
The intent is to provide a set of constraints that allow a variety of processors with different sized registers and memory spaces that can inter-operate with each other.  The presentation is a series of axioms, discussion and examples.   
Architectural constraints and feature sets are inferred from the axioms.   
  
1)        The first priority is low power in the sense of CoreMarks per joule.   
  
Discussion: This is not much of a constraint.  If anything it merely designates the current era of processor design.   It does suggest an evaluation metric.   
  
2)        The second priority is code density.   
  
Discussion: This constraint keeps the architecture from being purely an academic exercise.  The instruction encoding is not specified, merely, that it be efficient.  One way to increase code density is to use many small subroutines, e.g., highly factored code.  This approach to code density implies low overhead subroutine calls and returns.   
  
3)        The instruction set is open ended.   
  
Discussion: What this means is that if someone comes up with an instruction set, someone else can come up with a larger instruction set such that the first instruction set is a subset of the second.   
Example: There is single instruction format that is unbounded in size.  The larger instruction set merely takes more bits that the smaller.  There are several ways to indicate instruction length.  The simplest is to use one bit per instruction parcel to indicate the terminating parcel of the instruction.   
  
4)        20% of the instruction space is unallocated.   
  
Discussion: 20% is an arbitrary number less than 25% and greater than 12.5%.  This space is for custom instructions.  This applies for any length instruction.   
  
5)        There is an instruction or instruction sequence for in-line loading of any usage size immediate.   
  
Discussion: Although a convenience feature, it provides a standard way of handling any and all immediate data.  Otherwise, constant pool and constant pool pointer conventions are needed.   
  
6)        Register size is unspecified, except, that there be reasonable mechanism for doing calculations on data larger than register size.   
  
Discussion: The intent is that reduced size and complexity processors be possible that have the capability to emulate their larger "cousins".  The emulation may be via micro-code or subroutines.   
  
7)        The load/store instructions shall be able to efficiently load and store data that is shorter than the register size.   
  
Discussion: The intent is that data sizes are not "baked" into the hardware.  It should be reasonably efficient to choose data sizes not of the usual 8/16/32/64 sizes.   
  
8)        8-bit character addressing shall be supported.   
  
Discussion: Given the near universal acceptance of Unicode UTF-8, not to support it would be a great weakness.  One means of support is nine or twelve bit characters with one or four bits unused.   
  
Example: On a 24-bit word size architecture, a divide by three circuit could be used to convert 8-bit character addresses to word addresses.   
  
9)        Unsigned, signed and floating-point formats shall be defined.   
  
Discussion: This applies to data both larger and smaller than register size or memory word size.   
  
Example: Unsigned uses binary coding with the LSB at bit 0 and MSB at bit n-1. Signed uses sign-magnitude with the sign at bit 0 and MSB at bit n-1.  Floating uses sign-magnitude for both the mantissa and the exponent.  The mantissa is bit reversed such that the bit after the hidden MSB is at bit 2.  The mantissa and exponent bits interleave such that 8, 16, 32 and 64 bit floats match the exponent range of IEEE-754 floating-point.  The advantage of this scheme for signed and floating-point data is that zero extension can be used to load short data into a register and that truncation can be used for stores.  Means to prevent negative zero and/or trapping on negative zero should be provided.   
  
10)        Reasonable means shall be provided to do multi-precision arithmetic.   
  
Discussion: The intent is than convenience instructions exist for multi-precision arithmetic.   
  
Example: Each portion of a multi-precision signed number contains a sign bit.  Each portion of a multi-precision float contains an exponent.   
  
11)        It is preferred that the instruction set support both register and dual-stack based coding.   
  
Discussion: This is not a requirement as there is much argument.  As both are in wide use, the former for high performance and the later for interpreters, it makes sense that some reasonable accommodation be present.   
  
12)        It is required that code be position independent and thread base independent.   
  
Discussion: The intent is that there be program counter relative addressing and thread base relative addressing.  If there is only one thread, then thread base relative addressing is not needed.   
  
13)        A case instruction or instruction sequence shall be provided.   
  
Discussion: The intent is that dispatch through a table of code addresses or displacements be both safe and efficient.   
  
14)        A lookup table instruction or instruction sequence shall be provided.   
  
Discussion: This is the author's preference.  Small lookup tables are fast, efficient and direct means for implementing otherwise complicated logic.  The tables can be assumed to contain some power of two number of fields.   
  
15)        What is not specified:   
a.        Instruction parcel size   
b.        Instruction encoding   
c.        Register size   
d.        Memory word size   
e.        Data representation (although the author want the zero extension feature)   
f.        Number of registers and their allocation (general purpose or not)   
g.        Memory address size (presumably at least as large as register size)   
h.        The number of addressing modes   
i.        Signed and floating-point data formats must be defined and need not be implemented   
j.        The set of "baked in" data sizes is unspecified.  It is assumed that a variety of sizes are provided.   
k.        The choice of register, stack, accumulator or memory based architecture is not specified.  The preferred result is a hybrid.   
  
Discussion: The intent is that a diverse set of architectures be possible while maintaining some form of cross platform interoperability.  E.g., through subroutines or other means, equivalent functionality be possible when using high level languages.   
  
16)        Underlying assumptions:   
a.        Additional address and register adders are inexpensive.   
Discussion: The data load/store capabilities are made easier by having memory access to the calculated address and the calculated address plus one.  For example: loading a 36-bit data into a 48-bit register on an architecture having a 24-bit word size.  A second example is doing sign-magnitude arithmetic quickly by selecting the appropriate output from two complementary adders.   
b.        Support for load/store of non-power of two sized data is useful.   
Discussion: This is the key feature.  There are possible positive market drivers in cost sensitive markets for, say 24-bit addresses and/or 24-bit data.  Or one may want to give up some floating-point accuracy for other uses of a few bits from the mantissa.  Legacy applications abound.   
c.        The cost of a more elaborate instruction set is acceptable.   
d.        Binary representation.  Support for decimal formats considered a separate issue.   
e.        Virtual memory mapping considered a separate issue.   
f.        Absolute performance is not a dominate priority.     
  
End of file

**Ivan Godard Mar 2 2:44PM**

On 3/2/2015 11:50 AM, [jim.bra...@ieee.org](javascript:) wrote:   
> Partially as a response to Quadibloc & Skybuck Flying and partially as a result of my own interests the following is submitted for your review and comment.   
>  
> I would like nothing better than to be on an ad-hoc committee that meets in various exotic locations and argues the details.   
>  
> Introduction   
> The intent is to provide a set of constraints that allow a variety of processors with different sized registers and memory spaces that can inter-operate with each other.  The presentation is a series of axioms, discussion and examples.   
> Architectural constraints and feature sets are inferred from the axioms.   
>

Your list omits various constraints that appear to be assumed:   
-1) There are registers   
-2) Execution is imperative   
-3) Model is von Neuman   
  
Etc.   
  
These should be made explicit.

**Jim Brakefield Mar 2**

On Monday, March 2, 2015 at 2:44:12 PM UTC-6, Ivan Godard wrote:

- hide quoted text -

> On 3/2/2015 11:50 AM, jim.br...@ieee.org wrote:   
> > Partially as a response to Quadibloc & Skybuck Flying and partially as a result of my own interests the following is submitted for your review and comment.   
> >  
> > I would like nothing better than to be on an ad-hoc committee that meets in various exotic locations and argues the details.   
> >  
> > Introduction   
> > The intent is to provide a set of constraints that allow a variety of processors with different sized registers and memory spaces that can inter-operate with each other.  The presentation is a series of axioms, discussion and examples.   
> > Architectural constraints and feature sets are inferred from the axioms.   
> >  
>   
> Your list omits various constraints that appear to be assumed:   
> -1) There are registers   
> -2) Execution is imperative   
> -3) Model is von Neuman   
>   
> Etc.   
>   
> These should be made explicit.

OK, any more constraints will have numbers < -3 or > 16.  Until placed on the "belt" and renumbered.   
  
>> -1) There are registers   
Discussion: Don't know of a way to build a programmable computer without some registers.  At a minimum a PC and an accumulator.  If a serial machine, there are shift registers.  Tend to think of the Mill belt as a register file with the registers constantly being renumbered.  Somewhat the same situation as stack offsets.   
  
>> -2) Execution is imperative   
Discussion: Only exceptions I know of are "time-triggered" and "operands available" execution.  A new line of thought is to allow instructions to be allocated to registers and triggered by references to that register.   
A Prolog or Functional programming machine usually has an imperative engine underneath.  A pattern match and dispatch engine would also be non-imperative.

>> -3) Model is von Neuman

Discussion: Don't wish to exclude Harvard architecture, especially in its current form of instruction and data caches.  As for multiple PCs, that's more complexity than I wish to tackle.

**Stephen Fuld Mar 2**

On 3/2/2015 2:09 PM, [jim.bra...@ieee.org](javascript:) wrote:   
> On Monday, March 2, 2015 at 2:44:12 PM UTC-6, Ivan Godard wrote:   
>> On 3/2/2015 11:50 AM, jim.br...@ieee.org wrote:   
>>> Partially as a response to Quadibloc & Skybuck Flying and partially as a result of my own interests the following is submitted for your review and comment.   
>>>  
>>> I would like nothing better than to be on an ad-hoc committee that meets in various exotic locations and argues the details.   
>>>  
>>> Introduction   
>>> The intent is to provide a set of constraints that allow a variety of processors with different sized registers and memory spaces that can inter-operate with each other.  The presentation is a series of axioms, discussion and examples.   
>>> Architectural constraints and feature sets are inferred from the axioms.   
>>>  
>>  
>> Your list omits various constraints that appear to be assumed:   
>> -1) There are registers   
>> -2) Execution is imperative   
>> -3) Model is von Neuman   
>>  
>> Etc.   
>>  
>> These should be made explicit.   
>  
> OK, any more constraints will have numbers < -3 or > 16.  Until placed on the "belt" and renumbered.   
>  
>>> -1) There are registers   
> Discussion: Don't know of a way to build a programmable computer without some registers.  At a minimum a PC and an accumulator.

Well, you could fudge the PC by using a fixed memory address.  As for   
the accumulator, that is no problem.  See some older machines.  Think of   
each "operand" specification in an instruction being a memory address.   
After all, registers are merely a performance enhancement technique.  :-)

  If a serial machine, there are shift registers.  Tend to think of the   
Mill belt as a register file with the registers constantly being   
renumbered.  Somewhat the same situation as stack offsets.   
>  
>>> -2) Execution is imperative   
> Discussion: Only exceptions I know of are "time-triggered" and "operands available" execution.  A new line of thought is to allow instructions to be allocated to registers and triggered by references to that register.   
> A Prolog or Functional programming machine usually has an imperative engine underneath.  A pattern match and dispatch engine would also be non-imperative.

And a neural network type computer such as the ones IBM is developing.

>  
>>> -3) Model is von Neuman   
> Discussion: Don't wish to exclude Harvard architecture, especially in its current form of instruction and data caches.  As for multiple PCs, that's more complexity than I wish to tackle.   
>

--   
  - Stephen Fuld   
(e-mail address disguised to prevent spam)

**Robert…@yahoo.com Mar 2 6:21PM**

On Mon, 2 Mar 2015 14:09:05 -0800 (PST), [jim.bra...@ieee.org](javascript:)   
wrote:

>On Monday, March 2, 2015 at 2:44:12 PM UTC-6, Ivan Godard wrote:   
>> On 3/2/2015 11:50 AM, jim.br...@ieee.org wrote:   
>> > Partially as a response to Quadibloc & Skybuck Flying and partially as a result of my own interests the following is submitted for your review and comment.   
>> >  
>> > I would like nothing better than to be on an ad-hoc committee that meets in various exotic locations and argues the details.   
>> >  
>> > Introduction   
>> > The intent is to provide a set of constraints that allow a variety of processors with different sized registers and memory spaces that can inter-operate with each other.  The presentation is a series of axioms, discussion and examples.   
>> > Architectural constraints and feature sets are inferred from the axioms.   
>> >  
>>   
>> Your list omits various constraints that appear to be assumed:   
>> -1) There are registers   
>> -2) Execution is imperative   
>> -3) Model is von Neuman   
>>   
>> Etc.   
>>   
>> These should be made explicit.   
>  
>OK, any more constraints will have numbers < -3 or > 16.  Until placed on the "belt" and renumbered.   
>  
>>> -1) There are registers   
>Discussion: Don't know of a way to build a programmable computer without some registers.  At a minimum a PC and an accumulator.  If a serial machine, there are shift registers.  Tend to think of the Mill belt as a register file with the registers constantly being renumbered.  Somewhat the same situation as stack offsets.

Certainly there have been machines with where all operands were in   
storage.  It's still hard to avoid a PC on a machine executing serial   
instructions.   
  
In addition there have been a number of stack machines that also   
lacked data registers (but still had stack pointer registers).

**Jim Brakefield Mar 2**

On Monday, March 2, 2015 at 6:21:51 PM UTC-6, Stephen Fuld wrote:

- hide quoted text -

> On 3/2/2015 2:09 PM, jim.br...@ieee.org wrote:   
> > On Monday, March 2, 2015 at 2:44:12 PM UTC-6, Ivan Godard wrote:   
> >> On 3/2/2015 11:50 AM, jim.br...@ieee.org wrote:   
> >>> Partially as a response to Quadibloc & Skybuck Flying and partially as a result of my own interests the following is submitted for your review and comment.   
> >>>  
> >>> I would like nothing better than to be on an ad-hoc committee that meets in various exotic locations and argues the details.   
> >>>  
> >>> Introduction   
> >>> The intent is to provide a set of constraints that allow a variety of processors with different sized registers and memory spaces that can inter-operate with each other.  The presentation is a series of axioms, discussion and examples.   
> >>> Architectural constraints and feature sets are inferred from the axioms.   
> >>>  
> >>  
> >> Your list omits various constraints that appear to be assumed:   
> >> -1) There are registers   
> >> -2) Execution is imperative   
> >> -3) Model is von Neuman   
> >>  
> >> Etc.   
> >>  
> >> These should be made explicit.   
> >  
> > OK, any more constraints will have numbers < -3 or > 16.  Until placed on the "belt" and renumbered.   
> >  
> >>> -1) There are registers   
> > Discussion: Don't know of a way to build a programmable computer without some registers.  At a minimum a PC and an accumulator.   
>   
> Well, you could fudge the PC by using a fixed memory address.  As for   
> the accumulator, that is no problem.  See some older machines.  Think of   
> each "operand" specification in an instruction being a memory address.   
> After all, registers are merely a performance enhancement technique.  :-)   
>   
>   
>   If a serial machine, there are shift registers.  Tend to think of the   
> Mill belt as a register file with the registers constantly being   
> renumbered.  Somewhat the same situation as stack offsets.   
> >  
> >>> -2) Execution is imperative   
> > Discussion: Only exceptions I know of are "time-triggered" and "operands available" execution.  A new line of thought is to allow instructions to be allocated to registers and triggered by references to that register.   
> > A Prolog or Functional programming machine usually has an imperative engine underneath.  A pattern match and dispatch engine would also be non-imperative.   
>   
> And a neural network type computer such as the ones IBM is developing.   
>   
> >  
> >>> -3) Model is von Neuman   
> > Discussion: Don't wish to exclude Harvard architecture, especially in its current form of instruction and data caches.  As for multiple PCs, that's more complexity than I wish to tackle.   
> >  
>   
>   
> --   
>   - Stephen Fuld   
> (e-mail address disguised to prevent spam)   
  
>> Well, you could fudge the PC by using a fixed memory address.  As for   
>> the accumulator, that is no problem.  See some older machines.  Think of   
>> each "operand" specification in an instruction being a memory address.   
>> After all, registers are merely a performance enhancement technique.  :-)

Of course one could use a four address instruction and a three read/one write port memory and not have any registers.  It would, however, not meet the code density requirement.   
  
Or, one could keep all the "registers" at fixed memory locations and claim there no "real" registers (D flip-flops).

**Robert…@yahoo.com Mar 2**

On Mon, 2 Mar 2015 18:09:34 -0800 (PST), [jim.bra...@ieee.org](javascript:)

- hide quoted text -

wrote:   
  
>On Monday, March 2, 2015 at 6:21:51 PM UTC-6, Stephen Fuld wrote:   
>> On 3/2/2015 2:09 PM, jim.br...@ieee.org wrote:   
>> > On Monday, March 2, 2015 at 2:44:12 PM UTC-6, Ivan Godard wrote:   
>> >> On 3/2/2015 11:50 AM, jim.br...@ieee.org wrote:   
>> >>> Partially as a response to Quadibloc & Skybuck Flying and partially as a result of my own interests the following is submitted for your review and comment.   
>> >>>  
>> >>> I would like nothing better than to be on an ad-hoc committee that meets in various exotic locations and argues the details.   
>> >>>  
>> >>> Introduction   
>> >>> The intent is to provide a set of constraints that allow a variety of processors with different sized registers and memory spaces that can inter-operate with each other.  The presentation is a series of axioms, discussion and examples.   
>> >>> Architectural constraints and feature sets are inferred from the axioms.   
>> >>>  
>> >>  
>> >> Your list omits various constraints that appear to be assumed:   
>> >> -1) There are registers   
>> >> -2) Execution is imperative   
>> >> -3) Model is von Neuman   
>> >>  
>> >> Etc.   
>> >>  
>> >> These should be made explicit.   
>> >  
>> > OK, any more constraints will have numbers < -3 or > 16.  Until placed on the "belt" and renumbered.   
>> >  
>> >>> -1) There are registers   
>> > Discussion: Don't know of a way to build a programmable computer without some registers.  At a minimum a PC and an accumulator.   
>>   
>> Well, you could fudge the PC by using a fixed memory address.  As for   
>> the accumulator, that is no problem.  See some older machines.  Think of   
>> each "operand" specification in an instruction being a memory address.   
>> After all, registers are merely a performance enhancement technique.  :-)   
>>   
>>   
>>   If a serial machine, there are shift registers.  Tend to think of the   
>> Mill belt as a register file with the registers constantly being   
>> renumbered.  Somewhat the same situation as stack offsets.   
>> >  
>> >>> -2) Execution is imperative   
>> > Discussion: Only exceptions I know of are "time-triggered" and "operands available" execution.  A new line of thought is to allow instructions to be allocated to registers and triggered by references to that register.   
>> > A Prolog or Functional programming machine usually has an imperative engine underneath.  A pattern match and dispatch engine would also be non-imperative.   
>>   
>> And a neural network type computer such as the ones IBM is developing.   
>>   
>> >  
>> >>> -3) Model is von Neuman   
>> > Discussion: Don't wish to exclude Harvard architecture, especially in its current form of instruction and data caches.  As for multiple PCs, that's more complexity than I wish to tackle.   
>> >  
>>   
>>   
>> --   
>>   - Stephen Fuld   
>> (e-mail address disguised to prevent spam)   
>  
>>> Well, you could fudge the PC by using a fixed memory address.  As for   
>>> the accumulator, that is no problem.  See some older machines.  Think of   
>>> each "operand" specification in an instruction being a memory address.   
>>> After all, registers are merely a performance enhancement technique.  :-)   
>  
>Of course one could use a four address instruction and a three read/one write port memory and not have any registers.  It would, however, not meet the code density requirement.   
>  
>Or, one could keep all the "registers" at fixed memory locations and claim there no "real" registers (D flip-flops).

Almost what the TI990/TMS9900 did, except that they had a context   
register pointing to where in memory the register file was.  So   
context switches were quite fast.  Still they had a PC and status   
register.   
  
Some of the low-end PDP-11s stored their registers in memory   
locations, and you could update them by writing to those locations,   
although that was an implementation detail, not an architectural   
feature.

**Stephen Fuld Mar 3**

snip

>>> Well, you could fudge the PC by using a fixed memory address.  As for   
>>> the accumulator, that is no problem.  See some older machines.  Think of   
>>> each "operand" specification in an instruction being a memory address.   
>>> After all, registers are merely a performance enhancement technique.  :-)   
>  
> Of course one could use a four address instruction and a three read/one write port memory and not have any registers.

You only need two memory addresses in each instruction.  And if the PC   
is at a fixed memory address, it could be adjusted by the instruction   
length automatically by the hardware when an instruction is executed.  A   
branch is a move to the fixed address.

>  It would, however, not meet the code density requirement.

True, due to the longer instructions.  However, it does have the   
advantage of eliminating all load and store instructions.  And, for a   
small memory machine such that the maximum memory address is relatively   
few bits, it wasn't too bad.   
  
I'm not claiming this is practical today, just pointing out that a   
machine with no registers \*is\* conceivable.

- hide quoted text -

--   
  - Stephen Fuld   
(e-mail address disguised to prevent spam)

**David Brown Mar 3**

**Other recipients:**

There are many other computing models that are Turing equivalent, yet wildly different. Neural nets were mentioned by another poster. Cellular automation is another (which are very scalable, but difficult to program!), and of course there are the

- hide quoted text -

On 02/03/15 23:09, [jim.bra...@ieee.org](javascript:) wrote:   
> On Monday, March 2, 2015 at 2:44:12 PM UTC-6, Ivan Godard wrote:   
>> On 3/2/2015 11:50 AM, jim.br...@ieee.org wrote:   
>>> Partially as a response to Quadibloc & Skybuck Flying and   
>>> partially as a result of my own interests the following is   
>>> submitted for your review and comment.   
>>>   
>>> I would like nothing better than to be on an ad-hoc committee   
>>> that meets in various exotic locations and argues the details.   
>>>   
>>> Introduction The intent is to provide a set of constraints that   
>>> allow a variety of processors with different sized registers and   
>>> memory spaces that can inter-operate with each other.  The   
>>> presentation is a series of axioms, discussion and examples.   
>>> Architectural constraints and feature sets are inferred from the   
>>> axioms.   
>>>   
>>   
>> Your list omits various constraints that appear to be assumed: -1)   
>> There are registers -2) Execution is imperative -3) Model is von   
>> Neuman   
>>   
>> Etc.   
>>   
>> These should be made explicit.   
>   
> OK, any more constraints will have numbers < -3 or > 16.  Until   
> placed on the "belt" and renumbered.   
>   
>>> -1) There are registers   
> Discussion: Don't know of a way to build a programmable computer   
> without some registers.  At a minimum a PC and an accumulator.  If a   
> serial machine, there are shift registers.  Tend to think of the Mill   
> belt as a register file with the registers constantly being   
> renumbered.  Somewhat the same situation as stack offsets.   
>   
>>> -2) Execution is imperative   
> Discussion: Only exceptions I know of are "time-triggered" and   
> "operands available" execution.  A new line of thought is to allow   
> instructions to be allocated to registers and triggered by references   
> to that register. A Prolog or Functional programming machine usually   
> has an imperative engine underneath.  A pattern match and dispatch   
> engine would also be non-imperative.

There are many other computing models that are Turing equivalent, yet   
wildly different.  Neural nets were mentioned by another poster.   
Cellular automation is another (which are very scalable, but difficult   
to program!), and of course there are the modern fads such as quantum   
computation, DNA computation, etc.

>   
>>> -3) Model is von Neuman   
> Discussion: Don't wish to exclude Harvard architecture, especially in   
> its current form of instruction and data caches.  As for multiple   
> PCs, that's more complexity than I wish to tackle.   
>

Speaking as someone who has worked with Harvard architecture processors,   
I suggest you make life easier for programmers by sticking to von   
Neumann processors.  It's fine to have separate databuses or caches, but   
having separate logical address spaces for code and data is a big pain   
every time you want to access constant data.   
  
  
I would also suggest that you fix the format for signed and unsigned   
integers to use two's complement for signed integers, and a simple   
binary representation with no unused bits or codes.  There is nothing to   
lose by this - an implementation will not be easier or faster if it can   
use one's complement for signed integers (unlike for floating point,   
where there can be advantages in not using IEEE formats).

**Bruce Hoult Mar 3**

On Tuesday, March 3, 2015 at 4:58:36 PM UTC+13, robert...@yahoo.com wrote:   
> Some of the low-end PDP-11s stored their registers in memory   
> locations, and you could update them by writing to those locations,   
> although that was an implementation detail, not an architectural   
> feature.

A lot of ancient machine did that.   
  
Also the AVR 8 bit microcontrollers. The 32 8 bit registers are faster to access than memory, but they have memory addresses. In some low end models, the registers are the only RAM there is.   
  
Registers having addresses can be useful if you want to write a loop to operate on a few registers as an array e.g. to add a 32 or 64 bit integers stored in a number of 8 bit registers. (Of course if you have enough code space you'll probably unroll those loops)

**David Brown Mar 3**

**Other recipients:**

That's true in theory, but (on the AVR at least) it is useless in practice. Indirect access to memory (through pointer regisers) is so inefficient on the AVR that even for 64-bit operations (using two sets of 8 registers) it is not only much fast

- hide quoted text -

On 03/03/15 13:37, Bruce Hoult wrote:   
> On Tuesday, March 3, 2015 at 4:58:36 PM UTC+13, robert...@yahoo.com   
> wrote:   
>> Some of the low-end PDP-11s stored their registers in memory   
>> locations, and you could update them by writing to those   
>> locations, although that was an implementation detail, not an   
>> architectural feature.   
>   
> A lot of ancient machine did that.   
>   
> Also the AVR 8 bit microcontrollers. The 32 8 bit registers are   
> faster to access than memory, but they have memory addresses. In some   
> low end models, the registers are the only RAM there is.   
>   
> Registers having addresses can be useful if you want to write a loop   
> to operate on a few registers as an array e.g. to add a 32 or 64 bit   
> integers stored in a number of 8 bit registers. (Of course if you   
> have enough code space you'll probably unroll those loops)   
>

That's true in theory, but (on the AVR at least) it is useless in   
practice.  Indirect access to memory (through pointer regisers) is so   
inefficient on the AVR that even for 64-bit operations (using two sets   
of 8 registers) it is not only much faster, but probably takes less code   
space to user register-to-register operations rather than using the   
memory-mapped access to the registers.   
  
The only use I can think of for the memory mapped registers is if you   
want to save and restore the full bank of registers (for a context   
switch) - using indirect access could give a slightly smaller loop.  But   
even then, the difference is marginal, especially since the AVR does not   
support memory-memory or memory-stack operations.  (And any system   
supporting context switches is going to have a fair bit of code space.)   
 I have never seen memory-mapped registers used in practice.

**Peterf…@gmail.com Mar 3**

On Monday, March 2, 2015 at 11:09:12 PM UTC+1, jim.bra...@ieee.org wrote:   
> On Monday, March 2, 2015 at 2:44:12 PM UTC-6, Ivan Godard wrote:   
> > On 3/2/2015 11:50 AM, jim.br...@ieee.org wrote:   
> > > Partially as a response to Quadibloc & Skybuck Flying and partially as a result of my own interests the following is submitted for your review and comment.   
> > >  
> > > I would like nothing better than to be on an ad-hoc committee that meets in various exotic locations and argues the details.   
> > >  
> > > Introduction   
> > > The intent is to provide a set of constraints that allow a variety of processors with different sized registers and memory spaces that can inter-operate with each other.  The presentation is a series of axioms, discussion and examples.   
> > > Architectural constraints and feature sets are inferred from the axioms.   
> > >  
> >   
> > Your list omits various constraints that appear to be assumed:   
> > -1) There are registers   
> > -2) Execution is imperative   
> > -3) Model is von Neuman   
> >   
> > Etc.   
> >   
> > These should be made explicit.   
>   
> OK, any more constraints will have numbers < -3 or > 16.  Until placed on the "belt" and renumbered.   
>   
> >> -1) There are registers   
> Discussion: Don't know of a way to build a programmable computer without some registers.  At a minimum a PC and an accumulator.  If a serial machine, there are shift registers.  Tend to think of the Mill belt as a register file with the registers constantly being renumbered.  Somewhat the same situation as stack offsets.

What do you need an accumulator for?  Stack machines don't need it.   
  
You can also get around it and GPRs by referring backwards to previous instructions for the input values for the current instruction.   
  
You need some way, be it more or less explicit, to refer to previously computed values.   
  
You can use explicit value numbers during:   
 - both production and consumption ("registers")   
  
 - during consumption ("belt" and I think queues in general + reference to producing instruction as per an old posting in this group by Torben Mogensen)   
  
 - during neither ("stack" and "accumulator").  The latter requires some other way of referring to the non-default values through the insertion of stack manipulation instructions or spill/reload instruction (usually as load/store instructions that are indistinguishable from other load/store instructions).   
  
I guess explicit numbers during production is also an option but not one I've heard of in practice.   
  
Various "meta" instructions to renumber values are possible with all the above models (push/pop for accumulators and registers, dup/rot/drop/etc for stack machines, conform/rescue for the belt).  Their job is to keep the value numbers small -- in the extreme cases of accumulator/stack machines, to keep the number of values down to just /one/.   
  
If the value numbers are small, they can fit into instructions -- perhaps you can even fit more than one.  Or you can have more than one way to encode value numbers and you can choose the shorter one.   
  
The Mill uses an auto-aging scheme (aka autorenumbering) that automatically throws away single and zero use values and ensures that new values always have small numbers.  Small instructions (sorry, "operations") manipulate the value/value numbers to keep useful values live for a bit longer or to make the numbering scheme of different parts of a program fit together.   
  
It also uses a spill/restore scheme for longer-lived values that makes it clear that those read/write operations are different from normal load/store from/to memory.   
  
I don't know how to avoid a PC except by using very different computation models (van Wijngaarden grammars, single neuron neural networks with feedback, graph reduction, ...).  Maybe you can do something funky with TTA-architectures where certain magic destination addresses mean "please execute this value as several instructions".  Maybe coupled to some funky DMA-like streaming.  Come to think of it, the memory protection fault scheme of x86 is Turing complete:   
  
<https://news.ycombinator.com/item?id=5261598>   
  
and   
  
<http://www.cs.dartmouth.edu/~sws/pubs/bbss13.pdf>   
  
Which means that computation on cheap, widely accessible general-purpose computers WITHOUT (quite) using a PC is already here.   
  
-Peter

**Jim Brakefield Mar 3**

On Tuesday, March 3, 2015 at 5:33:58 AM UTC-6, David Brown wrote:

- hide quoted text -

> On 02/03/15 23:09, jim.br...@ieee.org wrote:   
> > On Monday, March 2, 2015 at 2:44:12 PM UTC-6, Ivan Godard wrote:   
> >> On 3/2/2015 11:50 AM, jim.br...@ieee.org wrote:   
> >>> Partially as a response to Quadibloc & Skybuck Flying and   
> >>> partially as a result of my own interests the following is   
> >>> submitted for your review and comment.   
> >>>   
> >>> I would like nothing better than to be on an ad-hoc committee   
> >>> that meets in various exotic locations and argues the details.   
> >>>   
> >>> Introduction The intent is to provide a set of constraints that   
> >>> allow a variety of processors with different sized registers and   
> >>> memory spaces that can inter-operate with each other.  The   
> >>> presentation is a series of axioms, discussion and examples.   
> >>> Architectural constraints and feature sets are inferred from the   
> >>> axioms.   
> >>>   
> >>   
> >> Your list omits various constraints that appear to be assumed: -1)   
> >> There are registers -2) Execution is imperative -3) Model is von   
> >> Neuman   
> >>   
> >> Etc.   
> >>   
> >> These should be made explicit.   
> >   
> > OK, any more constraints will have numbers < -3 or > 16.  Until   
> > placed on the "belt" and renumbered.   
> >   
> >>> -1) There are registers   
> > Discussion: Don't know of a way to build a programmable computer   
> > without some registers.  At a minimum a PC and an accumulator.  If a   
> > serial machine, there are shift registers.  Tend to think of the Mill   
> > belt as a register file with the registers constantly being   
> > renumbered.  Somewhat the same situation as stack offsets.   
> >   
> >>> -2) Execution is imperative   
> > Discussion: Only exceptions I know of are "time-triggered" and   
> > "operands available" execution.  A new line of thought is to allow   
> > instructions to be allocated to registers and triggered by references   
> > to that register. A Prolog or Functional programming machine usually   
> > has an imperative engine underneath.  A pattern match and dispatch   
> > engine would also be non-imperative.   
>   
> There are many other computing models that are Turing equivalent, yet   
> wildly different.  Neural nets were mentioned by another poster.   
> Cellular automation is another (which are very scalable, but difficult   
> to program!), and of course there are the modern fads such as quantum   
> computation, DNA computation, etc.   
>   
> >   
> >>> -3) Model is von Neuman   
> > Discussion: Don't wish to exclude Harvard architecture, especially in   
> > its current form of instruction and data caches.  As for multiple   
> > PCs, that's more complexity than I wish to tackle.   
> >   
>   
> Speaking as someone who has worked with Harvard architecture processors,   
> I suggest you make life easier for programmers by sticking to von   
> Neumann processors.  It's fine to have separate databuses or caches, but   
> having separate logical address spaces for code and data is a big pain   
> every time you want to access constant data.   
>   
>   
> I would also suggest that you fix the format for signed and unsigned   
> integers to use two's complement for signed integers, and a simple   
> binary representation with no unused bits or codes.  There is nothing to   
> lose by this - an implementation will not be easier or faster if it can   
> use one's complement for signed integers (unlike for floating point,   
> where there can be advantages in not using IEEE formats).

Would also prefer using 2's complement for negative numbers and offset binary for float exponent.  If one can associate some additional bits with each register, one could support both sign/magnitude and 2's complement with the ALU handling the reformatting/conversion.  This moves the reformat delay from the load path to the ALU which may or may not help.  Not a fan of one's complement, it has a feedback trap for the unwary.

**Quadibloc Mar 3**

On Monday, March 2, 2015 at 1:44:12 PM UTC-7, Ivan Godard wrote:   
  
> Your list omits various constraints that appear to be assumed:   
> -1) There are registers   
> -2) Execution is imperative   
> -3) Model is von Neuman

Those are not so much constraints as the \*absence\* of a constraint. At least in   
this specific context.   
  
The constrant that is absent? The design is not constrained to be highly   
original and innovative! Except for the specific innovations demanded by the   
spec - this way, it has more chance of being accepted and so on.   
  
So, yes, these conditions are constraints, but they are internal constraints   
within all but the exceptional genius who can take something different and make   
it work.   
  
John Savard

**Quadibloc Mar 3**

On Monday, March 2, 2015 at 3:09:12 PM UTC-7, jim.bra...@ieee.org wrote:   
  
> Discussion: Don't know of a way to build a programmable computer without some registers.  At a minimum a PC and an accumulator.

Well, one can cheat a little, and have just a program counter and a "workspace   
register" that points to sixteen "registers" in memory. The TI 9900.   
  
Or one can have a two-address machine that only has memory-memory instructions,   
so that all you need is a PC.   
  
But in today's world, using RAM instead of registers is hugely inefficient. The   
Mill avoids registers by routing outputs of functional units to the inputs of   
other functional units - there may be intrnal buffers to store values, but   
they're not addressed by name or number. Dataflow architectures are a   
better-known type of system that anticipated this.   
  
The point is, though, if one tries to do stuff like this, then finding a   
workable way to innovate like that becomes the whole project, and stuff like   
variable-size immediates become trivia in comparison.   
  
John Savard

**Quadibloc Mar 3**

On Monday, March 2, 2015 at 7:09:40 PM UTC-7, jim.bra...@ieee.org wrote:   
  
> Of course one could use a four address instruction

Three addresses would be enough to avoid both the PC and the accumulator. But why   
avoid the PC \*that\* way, unless there was a reason for the next instruction   
address not to be almost always the address after...   
  
like on a drum memory machine?   
  
John Savard

**Jacko Mar 3 1:49PM**

Some good ideas. I'd also add a RTL processor instruction which turns sequences of simple instructions into some of the more complicated instructions. The aim being to allow implementation of compilers and tools using a very simple subset of the ISA, and have hardware support via 1 useful RTL translate instruction to then improve the code generation process. This might be considered complicated, but as such an RTL processor could also do instruction reordering, the OoO execution overhead would be reduced to a one off load process against the CPU ID.   
  
Cheers Jacko

**Wolfgang kern Mar 3**

Quadibloc wrote:   
  
>> Of course one could use a four address instruction   
  
> Three addresses would be enough to avoid both the PC and the accumulator.   
> But why   
> avoid the PC \*that\* way, unless there was a reason for the next   
> instruction   
> address not to be almost always the address after...   
>  
> like on a drum memory machine?

I remember 8/16 bit MCs which had no PC nor an SP equivalent.   
It were this olde 1802/1804 MCUs and they could use any of the   
available 16 registers to become IP by a single byte instruction.   
  
They not even got CALL nor RET instructions, so it were up to the   
initial code to use two of the registers, one for CALL and another   
for return. And there were also the Zero-page memory...   
This (128 byte) were used to represent registers and a few other   
easy and fast accessible pointers to become IP or stack as desired.   
  
Perhaps we meanwhile lost interest in such a 'not so bad' idea.   
  
Why support abstraction when we can have fully truth ?   
\_\_   
wolfgang

**Jim Brakefield Mar 3**

On Tuesday, March 3, 2015 at 1:21:47 PM UTC-6, Quadibloc wrote:   
> On Monday, March 2, 2015 at 7:09:40 PM UTC-7, jim.bra...@ieee.org wrote:   
>   
> > Of course one could use a four address instruction   
>   
> Three addresses would be enough to avoid both the PC and the accumulator. But why   
> avoid the PC \*that\* way, unless there was a reason for the next instruction   
> address not to be almost always the address after...   
>   
> like on a drum memory machine?   
>   
> John Savard

Was trying to avoid both the PC AND the instruction register.  So a branch instruction needs two addresses for the next PC.  One can get by with three addresses if PC adr, source adr and 2nd source/destination adr.   
  
Registers versus memory locations:   
If all the registers are mapped into memory locations, then what you have is an emulator of a machine with registers.   
Should a register bank be considered memory? Is the glass half full or half empty?   
Gave a talk in 2003:   
<http://www.austin-cs.org/archive/Six-no-Eight-no-Eleven-Memories-for-Computer-Architecture>   
which looks at computer architecture from the point of view of memory mappings of the various components whether RAM, ROM, micro-code, register banks, stacks, memory mappings of logic, and anything else:   
                Abstract:   
By expanding the memory usage of the programmable computer into six regions it is shown that various architectures are in effect different multiplexings of these regions.  The historical reasons for particular multiplexings are usually economic.   
                Text:   
The six regions of memory are:   
Program memory                Where the programmer places his code   
Data memory                What the code operates on   
Micro-code memory        "Subroutines" beneath the assembler level   
Data stack                Where subroutine parameters & results go   
Return address stack        Where return addresses go   
Op-code lookup table        Maps op-codes into micro-code addresses   
  
...

**Jim Brakefield Mar 3**

On Tuesday, March 3, 2015 at 1:49:25 PM UTC-6, jacko wrote:   
> Some good ideas. I'd also add a RTL processor instruction which turns sequences of simple instructions into some of the more complicated instructions. The aim being to allow implementation of compilers and tools using a very simple subset of the ISA, and have hardware support via 1 useful RTL translate instruction to then improve the code generation process. This might be considered complicated, but as such an RTL processor could also do instruction reordering, the OoO execution overhead would be reduced to a one off load process against the CPU ID.   
>   
> Cheers Jacko

Can you explain further?   
  
One reason for the lookup table instruction is to use EDIF as an "assembly language": doing gate level or LUT level emulation.  For very slow logic, it is more efficient do logic emulation than to do FPGA logic.

**MitchAlsup Mar 3 6:08PM**

On Tuesday, March 3, 2015 at 7:40:47 PM UTC-6, jim.bra...@ieee.org wrote:   
  
> Was trying to avoid both the PC AND the instruction register.

In my considered opinion, this is like trying to make an aeroplane out of lead.

**Jim Brakefield Mar 3**

On Tuesday, March 3, 2015 at 8:06:08 PM UTC-6, MitchAlsup wrote:   
> On Tuesday, March 3, 2015 at 7:40:47 PM UTC-6, jim.bra...@ieee.org wrote:   
>   
> > Was trying to avoid both the PC AND the instruction register.     
>   
> In my considered opinion, this is like trying to make an aeroplane out of lead.

:) :) :) ...   
I know of cases where every effort was made to keep academics out of the picture.   
  
Seriously, I wish academics could lend insight into computer architecture.  In most cases they are historians.   
  
And even more seriously, in FPGAs there is an educational venue where a RISC processor has only one pipeline stage: instruction readout, decode, register readout, ALU operation and write back occur in a single clock.  Requires a PC register and any status register, but no other registers: chapter 7.3, Harris & Harris 2nd ed.

**Jim Brakefield Mar 3**

**Other recipients:**

Wish the dialogue was more about using constraints to define an (open?) architecture and not about the register count. I think this group has the maturity and experience to tackle the subject.

- hide quoted text -

On Tuesday, March 3, 2015 at 1:13:20 PM UTC-6, Quadibloc wrote:   
> On Monday, March 2, 2015 at 1:44:12 PM UTC-7, Ivan Godard wrote:   
>   
> > Your list omits various constraints that appear to be assumed:   
> > -1) There are registers   
> > -2) Execution is imperative   
> > -3) Model is von Neuman   
>   
> Those are not so much constraints as the \*absence\* of a constraint. At least in   
> this specific context.   
>   
> The constrant that is absent? The design is not constrained to be highly   
> original and innovative! Except for the specific innovations demanded by the   
> spec - this way, it has more chance of being accepted and so on.   
>   
> So, yes, these conditions are constraints, but they are internal constraints   
> within all but the exceptional genius who can take something different and make   
> it work.   
>   
> John Savard

Wish the dialogue was more about using constraints to define an (open?) architecture and not about the register count.  I think this group has the maturity and experience to tackle the subject.

**Bruce Hoult Mar 3**

**Other recipients:**

I think you'll find that the vast majority aeroplanes are constructed partially from lead, with with only the very slowest (or very sophisticated computerised) not having it. And if they don't have lead it's only because they use depleted uranium

- hide quoted text -

On Wednesday, March 4, 2015 at 3:06:08 PM UTC+13, MitchAlsup wrote:   
> On Tuesday, March 3, 2015 at 7:40:47 PM UTC-6, jim.bra...@ieee.org wrote:   
>   
> > Was trying to avoid both the PC AND the instruction register.     
>   
> In my considered opinion, this is like trying to make an aeroplane out of lead.

I think you'll find that the vast majority aeroplanes are constructed partially from lead, with with only the very slowest (or very sophisticated computerised) not having it.   
  
And if they don't have lead it's only because they use depleted uranium instead!   
  
For example, every Boeing jet airliner ever made uses such heavy metals (up to at least the 777 and 747-800), along with every McDonnell Douglas or Lockheed or BAC. The original 747 used depleted uranium (more recent ones are using tungsten, despite the extra expense). The original 737 used lead.   
  
Small and slow aircraft flying below about 80 knots may not have any lead. But even the humble Cessna 172 does.   
  
Certainly, such lead is not there as part of the load bearing structure. Rather, it's there to prevent the load bearing structure from breaking :-)

**Rob Warnock Mar 4 1:20AM**

**Other recipients:**

Bruce Hoult <bruce...@gmail.com> wrote: +---------------

Bruce Hoult  <[bruce...@gmail.com](javascript:)> wrote:   
+---------------

| Small and slow aircraft flying below about 80 knots   
| may not have any lead. But even the humble Cessna 172 does.

+---------------   
  
Actually, small aircraft are one of the few places   
[in the U.S., at least] where tetraethyl lead is   
still permitted as a gasoline additive, albeit 100LL   
[100 octane "low"-lead, which actually has more lead   
than leaded automotive gas did], see:   
  
    <http://en.wikipedia.org/wiki/Avgas#100LL_.28blue.29>   
  
Unfortunately, lead-free automobile gasoline ("mogas")   
is not a suitable replacement for leaded avgas, due to   
corrosion issues [from ethanol] and vapor-pressure issues   
[from the wider temperature and ambient pressures experienced   
in airplanes]. Work is ongoing to find an acceptable   
replacement for avgas.   
  
  
-Rob   
  
-----   
Rob Warnock                <[rp...@rpw3.org](javascript:)>  
627 26th Avenue                <<http://rpw3.org/>>  
San Mateo, CA 94403

**Bruce Hoult Mar 4 1:44AM**

On Wednesday, March 4, 2015 at 8:20:03 PM UTC+13, Rob Warnock wrote:   
> Bruce Hoult  <[bruce...@gmail.com](javascript:)> wrote:   
> +---------------   
> | Small and slow aircraft flying below about 80 knots   
> | may not have any lead. But even the humble Cessna 172 does.   
> +---------------   
>   
> Actually, small aircraft are one of the few places   
> [in the U.S., at least] where tetraethyl lead is   
> still permitted as a gasoline additive, albeit 100LL   
> [100 octane "low"-lead, which actually has more lead   
> than leaded automotive gas did], see:   
>   
>     <http://en.wikipedia.org/wiki/Avgas#100LL_.28blue.29>   
>   
> Unfortunately, lead-free automobile gasoline ("mogas")   
> is not a suitable replacement for leaded avgas, due to   
> corrosion issues [from ethanol] and vapor-pressure issues   
> [from the wider temperature and ambient pressures experienced   
> in airplanes]. Work is ongoing to find an acceptable   
> replacement for avgas.

I was referring to lead (or other heavy and cheap metals) used to adjust the centre of gravity of movable control surfaces to be in front of or at least near their hinge points, in order to suppress aerodynamic flutter.   
  
And not only control surfaces. The two outboard engines on the 747 have depleted uranium weights added inside the front of their cowling. Engines are mounted ahead of the wing pretty much purely to combat flutter of the wing in turbulence. If a bump causes the wing to deflect upwards, the heavy engine will make it twist down, decreasing the lift. And vice versa if a bump causes the wing to deflect downwards.

**MitchAlsup Mar 4 8:57AM**

On Wednesday, March 4, 2015 at 1:44:57 AM UTC-6, Bruce Hoult wrote:   
  
> I was referring to lead (or other heavy and cheap metals) used to adjust the centre of gravity of movable control surfaces to be in front of or at least near their hinge points, in order to suppress aerodynamic flutter.

I was talking about using lead as a structural metal in aeroplanes.

**Quadibloc Mar 4 11:25AM**

**Other recipients:**

You probably could use lead as the primary structural material of a small enough model airplane and still have the thing fly and not fall apart. John Savard

- hide quoted text -

On Tuesday, March 3, 2015 at 7:06:08 PM UTC-7, MitchAlsup wrote:   
> On Tuesday, March 3, 2015 at 7:40:47 PM UTC-6, jim.bra...@ieee.org wrote:   
>   
> > Was trying to avoid both the PC AND the instruction register.     
>   
> In my considered opinion, this is like trying to make an aeroplane out of lead.

You probably could use lead as the primary structural material of a small enough   
model airplane and still have the thing fly and not fall apart.   
  
John Savard

**Quadibloc Mar 4 11:28AM**

On Tuesday, March 3, 2015 at 6:40:47 PM UTC-7, jim.bra...@ieee.org wrote:   
> On Tuesday, March 3, 2015 at 1:21:47 PM UTC-6, Quadibloc wrote:   
> > On Monday, March 2, 2015 at 7:09:40 PM UTC-7, jim.bra...@ieee.org wrote:   
> >   
> > > Of course one could use a four address instruction   
> >   
> > Three addresses would be enough to avoid both the PC and the accumulator. But why   
> > avoid the PC \*that\* way, unless there was a reason for the next instruction   
> > address not to be almost always the address after...   
> >   
> > like on a drum memory machine?

> Was trying to avoid both the PC AND the instruction register.  So a branch   
> instruction needs two addresses for the next PC.  One can get by with three   
> addresses if PC adr, source adr and 2nd source/destination adr.

One could. But what earthly good would it do? It would just make the   
instructions longer for no particularly good reason.   
  
On the other hand, dataflow machines and the Mill derive, or at least purport   
to derive, a benefit from eliminating general registers, because data is used   
where it already is, instead of being put in registers and then taken out again.   
  
John Savard

**Quadibloc Mar 4 11:39AM**

On Tuesday, March 3, 2015 at 6:40:47 PM UTC-7, jim.bra...@ieee.org wrote:

> The six regions of memory are:   
> Program memory        Where the programmer places his code   
> Data memory           What the code operates on   
> Micro-code memory     "Subroutines" beneath the assembler level   
> Data stack            Where subroutine parameters & results go   
> Return address stack  Where return addresses go   
> Op-code lookup table  Maps op-codes into micro-code addresses

Of course, "micro-code memory" and the "op-code lookup table", while they can   
be implemented as ROM, can also be essentially nonexistent when a machine is   
hardwired.   
  
Then there's the case of the IBM 5100, or other machines with "nanocode" and   
"millicode" in addition to microcode.   
  
But then, you may have considered such issues, given the title of your talk.   
(And does one of these memories contain a fanatical devotion to the Pope?)   
  
Dataflow machines eliminate general registers in much the same way that a   
hardwired computer eliminates microcode memory. They're not replaced by a   
recognizable substitute, or by core locations. Instead, the programmer is   
tasked with seeing to it that the output of a calculation stage is forwarded as   
soon as it becomes available.   
  
In the Mill, though, there is the Belt. That is a storage area, and it does   
what registers or a stack would do; the idea is that it does it in a different   
way that is better.   
  
John Savard

**Jim Brakefield Mar 4 12:57PM**

On Wednesday, March 4, 2015 at 11:28:55 AM UTC-6, Quadibloc wrote:   
> On Tuesday, March 3, 2015 at 6:40:47 PM UTC-7, jim.bra...@ieee.org wrote:   
> > On Tuesday, March 3, 2015 at 1:21:47 PM UTC-6, Quadibloc wrote:   
> > > On Monday, March 2, 2015 at 7:09:40 PM UTC-7, jim.bra...@ieee.org wrote:   
> > >   
> > > > Of course one could use a four address instruction   
> > >   
> > > Three addresses would be enough to avoid both the PC and the accumulator. But why   
> > > avoid the PC \*that\* way, unless there was a reason for the next instruction   
> > > address not to be almost always the address after...   
> > >   
> > > like on a drum memory machine?   
>   
> > Was trying to avoid both the PC AND the instruction register.  So a branch   
> > instruction needs two addresses for the next PC.  One can get by with three   
> > addresses if PC adr, source adr and 2nd source/destination adr.   
>   
> One could. But what earthly good would it do? It would just make the   
> instructions longer for no particularly good reason.

> John Savard   
  
Have very little interest in a two, three or four address instruction set except when some or all of the addresses are implied.

**Jim Brakefield Mar 4 1:06PM**

**Other recipients:**

Going down the memory mapping path leads to several more architecture things that can be memory mapped. So what does one do when ROMs, micro-code, nano-code, etc. are mapped into logic? My answer is to map logic into memory via "numerizing" EDIF

- hide quoted text -

On Wednesday, March 4, 2015 at 11:39:26 AM UTC-6, Quadibloc wrote:   
> On Tuesday, March 3, 2015 at 6:40:47 PM UTC-7, jim.bra...@ieee.org wrote:   
>   
> > The six regions of memory are:   
> > Program memory        Where the programmer places his code   
> > Data memory           What the code operates on   
> > Micro-code memory     "Subroutines" beneath the assembler level   
> > Data stack            Where subroutine parameters & results go   
> > Return address stack  Where return addresses go   
> > Op-code lookup table  Maps op-codes into micro-code addresses   
>   
> Of course, "micro-code memory" and the "op-code lookup table", while they can   
> be implemented as ROM, can also be essentially nonexistent when a machine is   
> hardwired.   
>   
> Then there's the case of the IBM 5100, or other machines with "nanocode" and   
> "millicode" in addition to microcode.   
>   
> But then, you may have considered such issues, given the title of your talk.   
> (And does one of these memories contain a fanatical devotion to the Pope?)   
>   
> Dataflow machines eliminate general registers in much the same way that a   
> hardwired computer eliminates microcode memory. They're not replaced by a   
> recognizable substitute, or by core locations. Instead, the programmer is   
> tasked with seeing to it that the output of a calculation stage is forwarded as   
> soon as it becomes available.   
>   
> In the Mill, though, there is the Belt. That is a storage area, and it does   
> what registers or a stack would do; the idea is that it does it in a different   
> way that is better.   
>   
> John Savard   
  
>> Of course, "micro-code memory" and the "op-code lookup table", while they can   
>> be implemented as ROM, can also be essentially nonexistent when a machine is   
>> hardwired.

Going down the memory mapping path leads to several more architecture things that can be memory mapped.  So what does one do when ROMs, micro-code, nano-code, etc. are mapped into logic?  My answer is to map logic into memory via "numerizing" EDIF, e.g., mapping the net names and gate names into numbers and counting the total number of bits generated.  Doesn't help the cohesion of the idea of memory regions, does get one a upper limit on equivalent memory size.

**Ivan Godard Mar 4 2:52PM**

On 3/4/2015 9:39 AM, Quadibloc wrote:   
  
> In the Mill, though, there is the Belt. That is a storage area, and it does   
> what registers or a stack would do; the idea is that it does it in a different   
> way that is better.

A common, but erroneous, assumption.   
  
The belt is an abstraction, a namespace, without physical representation   
as wires and gates.   
  
To say that a value is "on the belt" is to say that it can be reached by   
belt temporal addressing. The value itself could be anywhere that has a   
datapath to the functional units.