The Open Channel is exactly what the name implies; a forum for the free exchange of technical ideas. Try to hold your contributions to one page maximum in the final magazine format (about 1000 words).
We'll accept anything (short of libel or obscenity) so long as it's submitted by a member of the Computer Society. If it's really bizarre we may require you to get another member to cosponsor your item.
Send everything to Jim Haynes, Applied Sciences, UC Santa Cruz, CA 95064.

The Rise and Fall of General-Purpose Registers
C. K. Yuen
University of Tasmania

For the assembly language programmer, the second half of the 60's was the age of the general-purpose registers. IBM gave him 16 of them in the 360, and he was free to use them for storing data, indexing and base addresses, or linking subroutines. To the "register deficient" programmer, long restricted to a diet of AC, MQ, and two index registers a la 7040, the 360 offered unprecedented freedom. A compute-bound program could now use most of the 16 registers to store and manipulate numbers, while a core-bound program could use most of them for addressing. Gone also was the need to perform address manipulation in one register (AC) and then load the result to another for memory access. Further, since every memory address had to be relative to a base register, 360 instructions needed only 12 bits for addressing. At the same time, object code was made statically relocatable—i.e., it could be loaded anywhere in core without change. All the programmer had to do to attain relocatability was set up enough base registers to point to all the program and data segments he wanted to access at any moment. Registers were allocated as base registers during assembling by the use of the pseudo-op USING, while the base addresses were loaded during execution by the use of a dummy subroutine call (branch and link). In fact, the trick of killing two birds with one stone looked so clever that it probably made IBM many converts.

Other manufacturers, however, were not all converted. The CDC 6000 computers had three sets of registers—eight data registers and 16 address registers separated into two divergent groups. Nothing general purpose about them! There was also a special base (relocation) register. The Univac 1108 did contain 128 registers that could be used for both data and indexing. However, it had special base registers (I-bank and D-bank bases). Burroughs went as far as making its registers undressable by implementing everything into a stack configuration. But the sheer size of the IBM market ensured that most programmers program general-purpose registers. Then, as the 60's drew to a close, the IBM of minicomputers, DEC, released the PDP-11 line. These had eight general-purpose registers, each of which could be used in eight addressing modes. They gave PDP-11 instructions a richness and flexibility never before imagined for minis. Vector access by the use of the auto-increment mode, stack operations by the auto-decrement and increment modes, and the automatic linkage of subroutine with main program by saving the return address in a linkage register (which can also be used to access parameters by the deferred auto-increment mode) are all impressive examples illustrating the power of general-purpose registers. Further, by using the PC as an index register, symbolic addresses are made automatically relocatable, though in general PDP-11 programs need relocation at load time. This, then, was the zenith of the general-purpose registers.

However, for IBM users the later 1960's also presented the age of multiprogramming, and the early 70's the age of virtual memory. Now, in order to ensure storage efficiency in a multiprogramming environment, it is desirable that programs be dynamically relocatable—i.e., they must be movable to new memory locations even after execution has started. This permits the compaction of free memory spaces to accommodate new programs that are too large to fit into any individual free blocks. Also, idle jobs can be rolled out and later reloaded into the new locations. But IBM 360 programs, unfortunately, are not dynamically relocatable. A job can be moved if all its base registers are updated to reflect the change in its starting address. The trouble is that the operating system has no means for identifying which registers are being used as base registers and which are used for data. In short, the general-purpose nature of the 360 registers is precisely the source of the relocation problem.

In theory, solutions can be found. If, for example, the setting up of a base register was by the use of a special load instruction, then the operating system would be able to maintain a running record of the base registers being used by each program. But here again, the problem turned out to be IBM's own cleverness: base registers are loaded by making dummy subroutine calls, and it would be rather more difficult (though not impossible) to distinguish between genuine and dummy branch-and-link executions. In any case, no such solutions were attempted. So when IBM introduced multiprogramming in the OS360/MFT system, fixed memory partitions were used. Vacant space in each partition was simply left unused, and jobs execute in fixed locations until completion. To some extent storage efficiency was improved in the OS360/MVT system, which permits some jobs to be specially designated as "rollable," so that memory compaction can be carried out by rolling out such jobs, though other programs still have to execute in fixed locations.

In the end, IBM found the solution in virtual memory. With this experience in mind, people often say that "virtual memory eliminates the need for relocation" since, it is claimed, "the virtual addresses accessed in a program do not change even though individual pages are rolled out and reloaded—all that one needs to do is to update the page table that translates the (unchanged) virtual addresses to the new physical addresses." This is, in fact, an imprecise statement: virtual memory as implemented by IBM does achieve this because it involves more than just paging: by itself, paging virtual memory does not solve the relocation difficulty—all it does is to transfer the problem from machine addresses to virtual addresses. To explain this, consider a system with 2^n virtual pages and 2^m machine pages. Each concurrent job is allocated a sufficiently large block in virtual space. In order words, it gets assigned contiguous virtual pages, even though the actual machine pages assigned to it need not be contiguous. As jobs terminate, their allotted virtual pages
are released and may be assigned to new jobs. However, since new jobs never match old ones exactly in size, some pages would be left vacant. So once again it would be desirable to relocate the jobs in virtual space to make the vacant pages contiguous again. The situation, in fact, is exactly analogous to the case of a physical memory with paging.

Thus, the 360/370 virtual memory had to use paged segmentation rather than just paging. In this, the virtual space consists not of 2^n continuous pages but of a number of variable size segments, each further divided into fixed-size pages. This permits the accommodation of different programs in different segments, and each program can start at address 0. A virtual address is given in three parts: the segment number, the page number, and the displacement within the page. Now we have a genuinely unchanging virtual address. A terminated program vacates a whole segment for allocation to a new job, without affecting other jobs/segments in any way. The need for relocation is now eliminated.

In this way, the general-purpose registers played a major part in influencing the direction of IBM hardware and operating systems development, forcing it to adopt more complex — and probably less efficient — solutions to subsequent problems. During the past decade or so, software costs have skyrocketed while hardware costs have tumbled, and it is not unfair to say that some of the blame could be heaped home to the general-purpose registers. After all, most of the world’s programmers use IBM 360/370 computers, and most of the world’s software was written for them. If they had been able to work with simpler software, then this would have substantially reduced programming cost, even though we could not quantify the difference. To a lesser extent we could say the same regarding DEC.

When it implemented virtual memory on the PDP-11/45 it had to use variable-size pages, with a maximum size of 8K bytes, and three-part virtual addresses.

It is intriguing to speculate what might have occurred if the 360 had followed a different design — say eight 32-bit data registers, and eight each of 24-bit index and base registers. IBM 360 programs would then be fully relocatable, since one needs only to update all eight base registers regardless of whether they are actually in use. (Presumably they would be unreadable, to discourage anyone from using them to store constants.) OS360/MFT would never have been developed, and OS360/MVT would have been rather more successful. Simple paging would have worked on the later 360’s and the 370’s. Also, since the R, X, and B fields of the instructions would now have only 3 bits each, more bits — say, 14 — could be used for address displacements. A base register value would then cover a larger memory segment, and one would not need to reload base registers quite so often. All in all, software development would have been easier. The extra cost in hardware would probably have been quite insignificant.

Thousands of 360/370/PDP-11 computers are still in use all around the world today, so it would appear that general-purpose registers are in no danger of extinction. Yet, one feels justified to say that history has passed them by. Like many social institutions, they started as forces of progress and enlightenment, only to end up as impediments to further progress and enlightenment. Current computing trends do not augur well for the future of general-purpose registers. The development of cache memory, for example, means that registers no longer provide a substantial speed advantage. The main justification for still having them is that they are fewer in number and require fewer bits to address. But this is precisely where separation of register types wins, since it keeps the number of each type small so that even fewer addressing bits are needed. May we say, the register is dead: long live the registers!

---

**Computers We Stock**

- IMSAI 599.
- SOL20 1095.
- Cromence 22 595.
- Appl Y 16K 1698.
- Compucolor 2750.
- Poly BB 735.
- Xitan I 769.
- Vector Graphics 619.
- Alpha Micro System 1495.

**Memory Modules We Stock**

- SSM MB7 200ns 16K 525
- Industrial µS Systems 8K 229
- SPACEBYTE 16K Static 599
- SSM MB7 450ns 8K 199
- Vector Graphics 250ns 8K 269

**DIP Switches**

- 1 10 100
- 4 1.85 1.65 1.45
- 5 1.85 1.65 1.45
- 6 1.85 1.65 1.45
- 7 2.00 1.80 1.60
- 8 2.20 1.90 1.70
- 9 2.30 2.10 1.75
- 10 2.40 2.20 1.80

---

**Essential Accoutrements**

**Texas Inst.**

Lo Profile sockets

<table>
<thead>
<tr>
<th>Pin</th>
<th>1</th>
<th>10</th>
<th>100*</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>.30</td>
<td>2.50</td>
<td>20.00</td>
</tr>
<tr>
<td>14</td>
<td>.25</td>
<td>2.00</td>
<td>18.00</td>
</tr>
<tr>
<td>16</td>
<td>.27</td>
<td>2.80</td>
<td>20.00</td>
</tr>
<tr>
<td>18</td>
<td>.40</td>
<td>3.20</td>
<td>27.00</td>
</tr>
<tr>
<td>20</td>
<td>6.00</td>
<td>6.00</td>
<td>40.00</td>
</tr>
<tr>
<td>22</td>
<td>5.00</td>
<td>4.00</td>
<td>30.00</td>
</tr>
<tr>
<td>24</td>
<td>5.00</td>
<td>4.00</td>
<td>30.00</td>
</tr>
<tr>
<td>26</td>
<td>5.00</td>
<td>4.00</td>
<td>30.00</td>
</tr>
<tr>
<td>40</td>
<td>5.00</td>
<td>4.00</td>
<td>30.00</td>
</tr>
</tbody>
</table>

*Write for 1K µp pricing

**Common DB Series Connector**

<table>
<thead>
<tr>
<th>1</th>
<th>10</th>
<th>100*</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB 9P</td>
<td>1.00</td>
<td>.80</td>
</tr>
<tr>
<td>DB 9S</td>
<td>1.50</td>
<td>1.40</td>
</tr>
<tr>
<td>DB15P</td>
<td>1.50</td>
<td>1.40</td>
</tr>
<tr>
<td>DB15S</td>
<td>2.25</td>
<td>2.00</td>
</tr>
<tr>
<td>DB15P</td>
<td>2.25</td>
<td>2.00</td>
</tr>
<tr>
<td>DB25S</td>
<td>3.25</td>
<td>3.10</td>
</tr>
<tr>
<td>DC37P</td>
<td>2.95</td>
<td>2.75</td>
</tr>
<tr>
<td>DC37S</td>
<td>4.90</td>
<td>4.50</td>
</tr>
<tr>
<td>DS50P</td>
<td>3.90</td>
<td>3.50</td>
</tr>
<tr>
<td>DS50S</td>
<td>6.50</td>
<td>6.00</td>
</tr>
</tbody>
</table>

We stock a complete line of 7400, 741S, 4000 CMOS

---

**Full ASCII Upper/Lower Case Computer Keyboards**

**Used Guaranteed Working**

Single Supply +5v @ 800 ma
Schematics Included
Basic Keyboard $45.00
Add: $5.00 for Upper Case Alpha
$10.00 for Numeric Keypad
$5.00 Misc. Function Switchs
$40.00 Metal Case w/bottom
$45.00 Metal with Walnut Ends
$1.50 Connector
$2.00 for 10 Extra Switches

---

**Computer Components**

5848 Sepulveda Blvd., Van Nuys, CA 91411 (213) 786-7411
4705 Artesia Blvd., Lawndale, CA 90260 (213) 370-4842

B of A and MC Welcome
Terms: Min order $10.00
add $2.00 P and H if order $25.00
Post Paid US if U. P.S. plus over $25.00
All orders U. S. Currency