**THE PDP-429**

The PDP-8 was a 12-bit machine; how can we expand it to a more conventional 16-bit machine? We considered some design issues in a [design document](http://www.cs.utexas.edu/users/peterson/pdp429/pdp429_design.html). What was the result?

**Memory**

Memory is 16-bit words. Memory addresses are 16-bit, word-addressable. This allows up to 65,536 16-bit words of memory.

Integers are represented in two's complement binary notation. There is no hardware support for floating point numbers. Characters are 8-bit ASCII.

**Registers**

All registers are 16 bits (except the Link bit). There are 4 general purpose registers: A, B, C, D.

The Link bit is one bit. It is never cleared by hardware. It is set whenever an arithmetic operation causes an overflow (ADD/SUB/Increment/Decrement/MUL) or a divide by zero (DIV).

In addition, there are 4 special purpose registers: stack pointer (SP), stack pointer limit (SPL), program counter (PC), and processor status word (PSW), The low order bit of the processor status word indicates if we are running or halted.

The stack grows down in memory; the stack pointer (SP) points to an empty location. Pushing to the stack stores in the word that the stack pointer points to, and then the stack pointer is decremented by one. Popping from the stack first increments the stack pointer by one, and then uses the word that the stack pointer points at. Errors include: (a) stack overflow, if SP < SPL when a stack push starts and (b) stack underflow, if SP wraps around to zero (SP = 0xFFFF when a stack pop starts).

The program counter (PC) is the address of the next instruction. After an instruction is fetched, the PC is incremented by one to point to the next instruction. A skip instruction may increment by one more. A jump, call or return instruction may reset the PC to another value. Arithmetic on the PC is modulo 16 bits, and does not cause overflow.

**Instructions**

There are 6 classes of instructions:

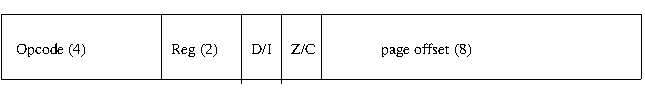
1. Non-register, Non-memory Instructions
2. Register Memory Reference Instructions
3. I/O Transfer Instructions
4. Non-register Memory Reference Instructions
5. Register-to-Register Instructions
6. Non-memory Register Instructions

**Non-register, Non-memory Instructions**

There are a small number of non-register, non-memory reference instructions. These instructions use the entire 16 bits as an opcode. The high-order 6 bits of these instructions are 000000.   
http://www.cs.utexas.edu/users/peterson/pdp429/nrnm_inst.gif

* 000000.0000000000 -- NOP.
* 000000.0000000001 -- HLT. The low-order bit of the PSW is set to 0; the machine halts.
* 000000.0000000010 -- RET. Pop the stack into the PC.

**Register Memory Reference Instructions**

This class of instructions specifies a 4-bit opcode, a 2-bit register selector, and a 10-bit memory address selector.   


The 2-bit register selector determines the general purpose register used in the instruction:

* 00 -- A register
* 01 -- B register
* 10 -- C register
* 11 -- D register

The memory address selector includes a zero/current page indicator (Z/C), a direct/indirect (D/I) indicator, and an 8-bit page offset. These are used to determine a memory address, just as with the PDP-8, except using pages of 256 words. The memory-operand is the contents of that memory address.

The opcodes are:

* 0001 -- ADD\*: the register + memory-operand -> Reg
* 0010 -- SUB\*: the register - memory-operand -> Reg
* 0011 -- MUL\*: the register \* memory-operand -> Reg
* 0100 -- DIV\*: the register / memory-operand -> Reg
* 0101 -- AND\*: the register & memory-operand -> Reg
* 0110 -- OR\*: the register | memory-operand -> Reg
* 0111 -- XOR\*: the register ^ memory-operand -> Reg
* 1000 -- LD\*: memory-operand -> Reg
* 1001 -- ST\*: the register -> memory-operand

The "\*" in the opcode indicates the register selected: ADDA, ADDB, ADDC, ADDD, and so on.

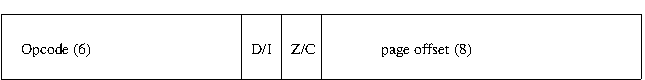
**I/O Transfer Instructions**

The IOT instruction has a 4-bit opcode of 1010, a 2-bit register selector, a 7-bit device number and a 3-bit function field.   
http://www.cs.utexas.edu/users/peterson/pdp429/iot_inst.gif

The 2-bit register selector is the same as for the Register Memory Reference Instructions.

* 1010 -- IOT\*: the register, function -> Device

**Non-register Memory Reference Instructions**

There are a small number of non-register memory reference instructions. These instructions combine the 4-bit opcode and 2-bit register fields of the Register Memory Reference Instructions to define a 6-bit opcode.   


The address of the memory-operand is computed the same as for the Register Memory Reference Instructions.

The opcodes are:

* 1011.00 -- ISZ: memory-operand + 1 -> memory-operand; if memory-operand == 0, PC + 1 -> PC
* 1011.01 -- JMP: address of memory-operand -> PC
* 1011.10 -- CALL: push return address (PC + 1) on stack; address of memory-operand -> PC
* 1100.00 -- PUSH: push memory-operand to the stack
* 1100.01 -- POP: pop top of stack and store in memory-operand.

**Register-to-Register Instructions**

The register-to-register instructions operate only on registers. A 3-bit register specifier is used to define a register. The register-to-register instructions are 3-operand instructions. Each instruction includes three 3-bit register specifiers (i,j,k) and define an instruction Rj op Rk -> Ri. The instruction specifies a 4-bit opcode of 1110 followed by a 3-bit sub-opcode, and three 3-bit register specifiers.   
http://www.cs.utexas.edu/users/peterson/pdp429/rr_inst.gif

The register specifiers are:

* 000 -- A register
* 001 -- B register
* 010 -- C register
* 011 -- D register
* 100 -- PC register (program counter)
* 101 -- PSW register (processor status word)
* 110 -- SP register (stack pointer)
* 111 -- SPL register (stack pointer limit)

The instructions are:

* 1110.000 -- MOD: Reg[j] % Reg[k] -> Reg[i]
* 1110.001 -- ADD: Reg[j] + Reg[k] -> Reg[i]
* 1110.010 -- SUB: Reg[j] - Reg[k] -> Reg[i]
* 1110.011 -- MUL: Reg[j] \* Reg[k] -> Reg[i]
* 1110.100 -- DIV: Reg[j] / Reg[k] -> Reg[i]
* 1110.101 -- AND: Reg[j] & Reg[k] -> Reg[i]
* 1110.110 -- OR: Reg[j] | Reg[k] -> Reg[i]
* 1110.111 -- XOR: Reg[j] ^ Reg[k] -> Reg[i]

**Non-memory Register Instructions**

This is like the operate instruction for the PDP-8.   
http://www.cs.utexas.edu/users/peterson/pdp429/nmr_inst.gif

The 2-bit register selector determines which general purpose register is used for this instruction. The encoding is the same as for the Register Memory Reference Instructions.

The individual bits for the Non-memory Register Instruction are:

* SM\* -- Skip if the register is negative (sign bit is 1)
* SZ\* -- Skip if the register is zero
* SNL -- Skip if the Link bit is non-zero
* RSS -- Reverse the Skip Sense
* CL\* -- Clear the register
* CLL -- Clear the Link bit
* CM\* -- Complement the register
* CML -- Complement the Link bit
* DC\* -- Decrement the register by one
* IN\* -- Increment the register by one

The "\*" in the opcode indicates the register selected: SMA, SMB, SMC, SMD, and so on.

The bits are evaluated in the order listed above.

**Instruction map by Numeric Opcode**

|  |  |  |
| --- | --- | --- |
| 0000.000000000000 | NOP | no operation |
| 0000.000000000001 | HLT | halt the processor |
| 0000.000000000010 | RET | pop stack -> PC |
| 0001.00 | ADDA | A-register + memory-operand -> A-register |
| 0001.01 | ADDB | B-register + memory-operand -> B-register |
| 0001.10 | ADDC | C-register + memory-operand -> C-register |
| 0001.11 | ADDD | D-register + memory-operand -> D-register |
| 0010.00 | SUBA | A-register - memory-operand -> A-register |
| 0010.01 | SUBB | B-register - memory-operand -> B-register |
| 0010.10 | SUBC | C-register - memory-operand -> C-register |
| 0010.11 | SUBD | D-register - memory-operand -> D-register |
| 0011.00 | MULA | A-register \* memory-operand -> A-register |
| 0011.01 | MULB | B-register \* memory-operand -> B-register |
| 0011.10 | MULC | C-register \* memory-operand -> C-register |
| 0011.11 | MULD | D-register \* memory-operand -> D-register |
| 0100.00 | DIVA | A-register / memory-operand -> A-register |
| 0100.01 | DIVB | B-register / memory-operand -> B-register |
| 0100.10 | DIVC | C-register / memory-operand -> C-register |
| 0100.11 | DIVD | D-register / memory-operand -> D-register |
| 0101.00 | ANDA | A-register & memory-operand -> A-register |
| 0101.01 | ANDB | B-register & memory-operand -> B-register |
| 0101.10 | ANDC | C-register & memory-operand -> C-register |
| 0101.11 | ANDD | D-register & memory-operand -> D-register |
| 0110.00 | ORA | A-register | memory-operand -> A-register |
| 0110.01 | ORB | B-register | memory-operand -> B-register |
| 0110.10 | ORC | C-register | memory-operand -> C-register |
| 0110.11 | ORD | D-register | memory-operand -> D-register |
| 0111.00 | XORA | A-register ^ memory-operand -> A-register |
| 0111.01 | XORB | B-register ^ memory-operand -> B-register |
| 0111.10 | XORC | C-register ^ memory-operand -> C-register |
| 0111.11 | XORD | D-register ^ memory-operand -> D-register |
| 1000.00 | LDA | memory-operand -> A-register |
| 1000.01 | LDB | memory-operand -> B-register |
| 1000.10 | LDC | memory-operand -> C-register |
| 1000.11 | LDD | memory-operand -> D-register |
| 1001.00 | STA | A-register -> memory-operand |
| 1001.01 | STB | B-register -> memory-operand |
| 1001.10 | STC | C-register -> memory-operand |
| 1001.11 | STD | D-register -> memory-operand |
| 1010.00 | IOTA | A-register, function -> Device |
| 1010.01 | IOTB | B-register, function -> Device |
| 1010.10 | IOTC | C-register, function -> Device |
| 1010.11 | IOTD | D-register, function -> Device |
| 1011.00 | ISZ | memory-operand + 1 -> memory-operand; if memory-operand == 0, PC + 1 -> PC |
| 1011.01 | JMP | address of memory-operand -> PC |
| 1011.10 | CALL | push return address (PC + 1) on stack; address of memory-operand -> PC |
| 1100.00 | PUSH | push memory-operand to the stack |
| 1100.01 | POP | pop top of stack and store in memory-operand. |
| 1110.000 | MOD | Reg[j] % Reg[k] -> Reg[i] |
| 1110.001 | ADD | Reg[j] + Reg[k] -> Reg[i] |
| 1110.010 | SUB | Reg[j] - Reg[k] -> Reg[i] |
| 1110.011 | MUL | Reg[j] \* Reg[k] -> Reg[i] |
| 1110.100 | DIV | Reg[j] / Reg[k] -> Reg[i] |
| 1110.101 | AND | Reg[j] & Reg[k] -> Reg[i] |
| 1110.110 | OR | Reg[j] | Reg[k] -> Reg[i] |
| 1110.111 | XOR | Reg[j] ^ Reg[k] -> Reg[i] |
| 1111.00.1000000000 | SMA | Skip if the register is negative (sign bit is 1) |
| 1111.01.1000000000 | SMB | Skip if the register is negative (sign bit is 1) |
| 1111.10.1000000000 | SMC | Skip if the register is negative (sign bit is 1) |
| 1111.11.1000000000 | SMD | Skip if the register is negative (sign bit is 1) |
| 1111.00.0100000000 | SZA | Skip if the register is zero |
| 1111.01.0100000000 | SZB | Skip if the register is zero |
| 1111.10.0100000000 | SZC | Skip if the register is zero |
| 1111.11.0100000000 | SZD | Skip if the register is zero |
| 1111.00.0010000000 | SNL | Skip if the Link bit is non-zero |
| 1111.00.0001000000 | RSS | Reverse the Skip Sense |
| 1111.00.0000100000 | CLA | Clear the register |
| 1111.01.0000100000 | CLB | Clear the register |
| 1111.10.0000100000 | CLC | Clear the register |
| 1111.11.0000100000 | CLD | Clear the register |
| 1111.00.0000010000 | CLL | Clear the Link bit |
| 1111.00.0000001000 | CMA | Complement the register |
| 1111.01.0000001000 | CMB | Complement the register |
| 1111.10.0000001000 | CMC | Complement the register |
| 1111.11.0000001000 | CMD | Complement the register |
| 1111.00.0000000100 | CML | Complement the Link bit |
| 1111.00.0000000010 | DCA | Decrement the register by one |
| 1111.01.0000000010 | DCB | Decrement the register by one |
| 1111.10.0000000010 | DCC | Decrement the register by one |
| 1111.11.0000000010 | DCD | Decrement the register by one |
| 1111.00.0000000001 | INA | Increment the register by one |
| 1111.01.0000000001 | INB | Increment the register by one |
| 1111.10.0000000001 | INC | Increment the register by one |
| 1111.11.0000000001 | IND | Increment the register by one |

**Instruction map by Symbolic Opcode**

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
| ADD | 1110.001 | Reg[j] + Reg[k] -> Reg[i] |
| ADDA | 0001.00 | A-register + memory-operand -> A-register |
| ADDB | 0001.01 | B-register + memory-operand -> B-register |
| ADDC | 0001.10 | C-register + memory-operand -> C-register |
| ADDD | 0001.11 | D-register + memory-operand -> D-register |
| AND | 1110.101 | Reg[j] & Reg[k] -> Reg[i] |
| ANDA | 0101.00 | A-register & memory-operand -> A-register |
| ANDB | 0101.01 | B-register & memory-operand -> B-register |
| ANDC | 0101.10 | C-register & memory-operand -> C-register |
| ANDD | 0101.11 | D-register & memory-operand -> D-register |
| CALL | 1011.10 | push return address (PC + 1) on stack; address of memory-operand -> PC |
| CLA | 1111.00.0000100000 | Clear A-register |
| CLB | 1111.01.0000100000 | Clear B-register |
| CLC | 1111.10.0000100000 | Clear C-register |
| CLD | 1111.11.0000100000 | Clear D-register |
| CLL | 1111.00.0000010000 | Clear the Link bit |
| CMA | 1111.00.0000001000 | Complement A-register |
| CMB | 1111.01.0000001000 | Complement B-register |
| CMC | 1111.10.0000001000 | Complement C-register |
| CMD | 1111.11.0000001000 | Complement D-register |
| CML | 1111.00.0000000100 | Complement the Link bit |
| DCA | 1111.00.0000000010 | Decrement A-register by one |
| DCB | 1111.01.0000000010 | Decrement B-register by one |
| DCC | 1111.10.0000000010 | Decrement C-register by one |
| DCD | 1111.11.0000000010 | Decrement D-register by one |
| DIV | 1110.100 | Reg[j] / Reg[k] -> Reg[i] |
| DIVA | 0100.00 | A-register / memory-operand -> A-register |
| DIVB | 0100.01 | B-register / memory-operand -> B-register |
| DIVC | 0100.10 | C-register / memory-operand -> C-register |
| DIVD | 0100.11 | D-register / memory-operand -> D-register |
| HLT | 0000.000000000001 |  |
| INA | 1111.00.0000000001 | Increment A-register by one |
| INB | 1111.01.0000000001 | Increment B-register by one |
| INC | 1111.10.0000000001 | Increment C-register by one |
| IND | 1111.11.0000000001 | Increment D-register by one |
| IOTA | 1010.00 | A-register, function -> Device |
| IOTB | 1010.01 | B-register, function -> Device |
| IOTC | 1010.10 | C-register, function -> Device |
| IOTD | 1010.11 | D-register, function -> Device |
| ISZ | 1011.00 | memory-operand + 1 -> memory-operand; if memory-operand == 0, PC + 1 -> PC |
| JMP | 1011.01 | address of memory-operand -> PC |
| LDA | 1000.00 | memory-operand -> A-register |
| LDB | 1000.01 | memory-operand -> B-register |
| LDC | 1000.10 | memory-operand -> C-register |
| LDD | 1000.11 | memory-operand -> D-register |
| MOD | 1110.000 | Reg[j] % Reg[k] -> Reg[i] |
| MUL | 1110.011 | Reg[j] \* Reg[k] -> Reg[i] |
| MULA | 0011.00 | A-register \* memory-operand -> A-register |
| MULB | 0011.01 | B-register \* memory-operand -> B-register |
| MULC | 0011.10 | C-register \* memory-operand -> C-register |
| MULD | 0011.11 | D-register \* memory-operand -> D-register |
| NOP | 0000.000000000000 |  |
| OR | 1110.110 | Reg[j] | Reg[k] -> Reg[i] |
| ORA | 0110.00 | A-register | memory-operand -> A-register |
| ORB | 0110.01 | B-register | memory-operand -> B-register |
| ORC | 0110.10 | C-register | memory-operand -> C-register |
| ORD | 0110.11 | D-register | memory-operand -> D-register |
| POP | 1100.01 | pop top of stack and store in memory-operand. |
| PUSH | 1100.00 | push memory-operand to the stack |
| RET | 0000.000000000010 | pop stack -> PC |
| RSS | 1111.00.0001000000 | Reverse the Skip Sense |
| SMA | 1111.00.1000000000 | Skip if A-register is negative (sign bit is 1) |
| SMB | 1111.01.1000000000 | Skip if B-register is negative (sign bit is 1) |
| SMC | 1111.10.1000000000 | Skip if C-register is negative (sign bit is 1) |
| SMD | 1111.11.1000000000 | Skip if D-register is negative (sign bit is 1) |
| SNL | 1111.00.0010000000 | Skip if the Link bit is non-zero |
| STA | 1001.00 | A-register -> memory-operand |
| STB | 1001.01 | B-register -> memory-operand |
| STC | 1001.10 | C-register -> memory-operand |
| STD | 1001.11 | D-register -> memory-operand |
| SUB | 1110.010 | Reg[j] - Reg[k] -> Reg[i] |
| SUBA | 0010.00 | A-register - memory-operand -> A-register |
| SUBB | 0010.01 | B-register - memory-operand -> B-register |
| SUBC | 0010.10 | C-register - memory-operand -> C-register |
| SUBD | 0010.11 | D-register - memory-operand -> D-register |
| SZA | 1111.00.0100000000 | Skip if A-register is zero |
| SZB | 1111.01.0100000000 | Skip if B-register is zero |
| SZC | 1111.10.0100000000 | Skip if C-register is zero |
| SZD | 1111.11.0100000000 | Skip if D-register is zero |
| XOR | 1110.111 | Reg[j] ^ Reg[k] -> Reg[i] |
| XORA | 0111.00 | A-register ^ memory-operand -> A-register |
| XORB | 0111.01 | B-register ^ memory-operand -> B-register |
| XORC | 0111.10 | C-register ^ memory-operand -> C-register |
| XORD | 0111.11 | D-register ^ memory-operand -> D-register |