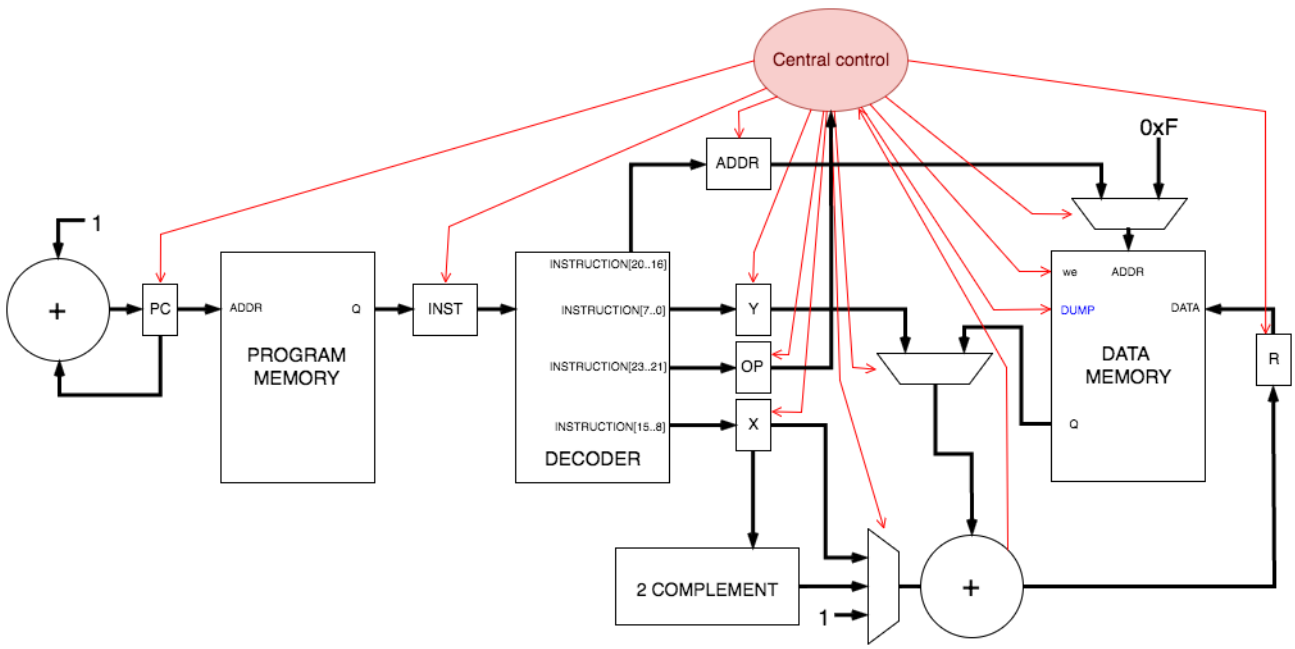
# Projeto Nibble v0.1

O projeto consiste em implementar um processador de 8 bits com conjunto de instruções reduzido, cujo núcleo foi projetado com arquitetura Harvard e 4 estágios de pipeline. O nome do projeto é Nibble, figura abaixo, o qual possui apenas 6 instruções: ADD, ADDP, SUB, SUBP, END e MDUMP. As instruções são gravadas na memória de programa (também chamada de flash) e utiliza a memória RAM para armazenar os resultados das operações. O conteúdo do processador são os seguintes módulos:

1. Memória de programa (ROM);
2. Decodificador de instrução;
3. Memória de dados (RAM);
4. Multiplexadores;
5. Módulo complemento a 2;
6. Somador completo (8 bits);
7. Registradores de carga paralela [1] (PC, ADDR, X, OP, Y e R) ;
8. Módulo de controle central.




**Observação**: Os módulos 1 e 3 serão fornecidos, não precisam ser desenvolvidos. Os outros módulos ficam a critério do aluno.

## Instruções

O processador Nibble possui 6 instrução: ADD, ADDP, SUB, SUBP, END e MDUMP. Todas as instruções tem o mesmo tamanho 24 bits. Onde 3 são para código de operação e os outros 21 bits para operandos ou reservados. A descrição da cada instrução está detalhada nas tabelas a seguir. **Atenção**: em todos os exemplos são usados números binário e hexadecimal.

## ADD

| Operação | Endereço de armazenamento | Literal | Literal |
|---|---|---|---|
| ADD | addr | x | y |
| A operação realiza a soma dos valores de ***x*** e ***y*** e armazena o resultado no endereço ***addr*** da memória RAM.<br><br>***ADD addr, x, y => [addr] = x + y*** | | | |
| 000 | aaaaa(5 bits) | xxxxxxxx(8 bits) | yyyyyyyy(8 bits) |
| **Exemplo:**<br>000.00110.00000111.00000001 <=> ADD 6, 7, 1<br><br>O endereço 0x6 da memória RAM receberá o valor 0x8. | | | |

## ADDP

| Operação | Endereço de armazenamento | Literal | Reservado |
|---|---|---|---|
| ADDP | addr | x | - |
| A operação realiza a soma dos valores e ***x*** e do conteúdo do endereço ***addr*** e armazena o resultado no endereço ***addr*** da memória RAM.<br><br>***ADDP addr, y => [addr] = [addr] + x*** | | | |
| 001 | aaaaa(5 bits) | xxxxxxxx(8 bits) | - - - - - - - - (8 bits) |
| **Exemplo:**<br>001.00010.00001001.00000000 <=> ADDP 2, 9<br><br>O conteúdo do endereço 0x2 da memória RAM será incrementado em 9. | | | |

## SUB

| Operação | Endereço de armazenamento | Literal | Literal |
|---|---|---|---|
| SUB | addr | x | y |
| A operação subtrai ***x*** de ***y*** e armazena o resultado no endereço ***addr*** da memória RAM.<br><br>***SUB addr, y, x => addr = y - x*** | | | |
| 010 | aaaaa(5 bits) | xxxxxxxx(8 bits) | yyyyyyyy(8 bits) |
| **Exemplo:**<br>010.01010.00000010.00001001 <=> SUB A, 2, 9 | | | |

| |
|---|
| O conteúdo do endereço 0xA da memória RAM receberá o valor 7. |

### SUBP

| Operação | Endereço de armazenamento | Literal | Reservado |
|---|---|---|---|
| SUBP | addr | x | - |
| A operação subtrai ***x*** do conteúdo de ***addr*** e armazena o resultado no endereço ***addr*** da memória RAM.<br><br>***SUBP addr, y => [addr] = [addr] - x*** | | | |
| 011 | aaaaa(5 bits) | xxxxxxxx(8 bits) | - - - - - - - - (8 bits) |
| **Exemplo:**<br>011.00010.00001001.00000000 <=> SUBP 2, 9<br><br>O conteúdo do endereço 0x2 da memória RAM será decrementado em 9 unidades. | | | |

**Observação**: Todas as operações aritméticas podem gerar a exceção overflow. Caso isso ocorra, o somador coloca a saída ovf para o valor 1 indicando ao controle central o erro. A posição 0xF da memória RAM armazena um contador de overflow ocorridos durante a execução. Para cada overflow que ocorrer, o contador deve ser incrementado na posição 0xF da memória RAM.

### END

| Operação | Reservado |
|---|---|
| END | - |
| Essa instrução indica ao processador que o programa acabou. | |
| 100 | - - - - - - - - - - - - - - - - - - - - - (21 bits) |
| **Exemplo:**<br>100.00000.00000000.00000000 <=> END<br><br>O Controle Central para de funcionar. | |

### MDUMP

| Operação | Reservado |
|---|---|
| MDUMP | - |
| Ferramenta de depuração. Ao ser executada essa operação, o conteúdo da memória RAM será externalizado um a um na porta Q da memória. A implementação dessa instrução já está contemplada no código da memória RAM, basta o controle central indicar quando executar essa operação colocando o valor da entrada DUMP na memória para 1. Caso contrário o valor da entrada DUMP deve ser mantido em 0. | |
| 111 | - - - - - - - - - - - - - - - - - - - - - (21 bits) |

**Exemplo:**
111.00000.00000000.00000000 <=> MDUMP

# Módulos

Os módulos do processador estão descritos a seguir. Os módulos Memória ROM e Memória RAM serão fornecidos.

## Somador completo[1]

Módulo que implementa um somador completo de oito bits o qual contém detector de overflow. Caso haja um overflow a saída ovf é mantida um até que os operandos sejam alterados.

## Complemento a 2[1]

Esse módulo é responsável por fazer o complemento a 2 de um número de 8 bits que ele recebe na entrada. A do módulo saída tem 8 bits.

## Decodificador de instruções[1]

O módulo decodificador é responsável por separar o conteúdo da instrução em 4 partes:

- Código de operação -> ***OP[3..0] = INST[23..21]***
- Endereço -> ***ADDR[5..0] = INST[20..15]***
- Operando de 8 bits -> ***X[7..0] = INST[15..8]***
- Operando de 8 bits -> ***Y[7..0] = INST[7..0]***

## Registradores de carga paralela

Os registradores PC, INST, ADDR, Y, X, OP, X e R são registradores de carga paralela [1]. Eles possuem as seguinte entradas e saídas: data_in (8 bits), clk (1 bit), en (1 bit) e data_out (8bits). A mudança do dado de saída é engatilhada na subida do clock. O valor da saída apenas será alterado se o pino en (*enable*) estiver habilitado; caso contrário, o valor da saída se mantém independente das mudanças do clock.

## Memória de programa

A memória de programa é somente leitura e contém o programa a ser executado pelo processador. Possui palavras de 24 bits, uma entrada ***ADDR*** de 8 bits e uma saída ***Q*** onde será externalizado o conteúdo (24 bits) do endereço colocado na entrada. O dado de saída será alterado quando o endereço de entrada for alterado. A mudança da saída é engatilhada na descida do clock.

## Memória de dados

A memória de programa é de leitura e escrita. Tem como entrada o ***ADDR*** 5 bits (endereço de leitura/escrita), a flag ***we*** 1 bit (*write_enable*), a flag ***DUMP*** 1 bit reservada e a entrada ***DATA*** a qual recebe o valor a ser escrito na memória. A leitura é feita através da saída Q de 8 bits a qual é engatilhada na descida do clock e apenas será alterada caso a flag ***we*** esteja desabilitada e a entrada ***ADDR*** mude. Para escrita (engatilhada na descida do clock) o dado deve ser colocado na entrada ***DATA***, o endereço de escrita deve ser colocado no ***ADDR*** e a flag ***we*** deve ser habilitada.

## Controle central

É o módulo responsável pelo funcionamento do sistema. Ele habilita os registradores, ajusta os multiplexadores e utiliza as memórias de forma harmônica. O Funcionamento do Controle Central é baseado no conceito de pipeline onde podemos identificar 3 estágio independentes de funcionamento do processador: FETCH, DECODE, EXECUTE/STORE.

1. FETCH: é quando o valor de PC é alterado e o conteúdo da memória de programa (instrução) fica disponível;
2. DECODE: o registrador de instrução (INST) é alterado deixando pronta a instrução decodificada;
3. EXECUTE: quando os registradores ADDR, Y, OP e X recebem os dados da instrução e o controle central ajusta os multiplexadores para que a instrução seja executada de forma apropriada. Em algumas instruções deve-se ter acesso à memória antes de executar, isso pode leva o tempo para a memória externalizar o dado (lembre-se que a memória, ao contrário dos registradores, é engatilhada na descida). O resultado fica disponível para ser armazenado no registrador R para posterior armazenamento na memória;
4. STORE: os dados do registrador R serão armazenados na memória de dados. Observe que o processador só poderá executar uma nova instrução quando o STORE terminar sua operação.

*1 - Esse módulo apresenta apenas lógica combinacional.*

# Referências

[1] - https://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Seq/parload.html
[2] - https://www.altera.com/support/training/university/materials-tutorials.html