É difícil comparar um chip CISC (como o Core i7) e um chip RISC (como o OMAP4430) apenas com base na velocidade do *clock*. Por exemplo, os dois núcleos ARM A9 no OMAP4430 têm uma velocidade máxima de execução de quatro instruções por ciclo de *clock*, dando-lhe quase a mesma taxa de execução dos processadores superescalares de largura 4 do Core i7. Entretanto, o Core i7 alcança execução de programa mais rápida, pois tem até seis processadores rodando com uma velocidade de *clock* 3,5 vezes mais rápida (3,5 GHz) que o OMAP4430. O OMAP4430 pode parecer uma tartaruga correndo ao lado da lebre do Core i7, mas a tartaruga usa muito menos potência, e pode terminar primeiro, ainda mais se a bateria da lebre não for muito grande.

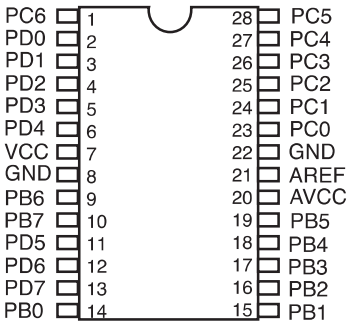
3.5.3 O microcontrolador Atmel ATmega168

Tanto o Core i7 quanto a OMAP4430 são exemplos de CPUs de alto desempenho projetadas para construir dispositivos de computação altamente eficazes, com o Core i7 voltado para aplicações de desktop enquanto o OMAP4430 é voltado para aplicações móveis. Quando pensam em computadores, são esses os tipos de sistemas que muitas pessoas têm em mente. Entretanto, existe todo outro universo de computadores que na verdade é muito maior: sistemas embutidos. Nesta seção, vamos examinar brevemente esse outro universo.

Talvez não seja um grande exagero dizer que todo equipamento elétrico que custe mais de 100 dólares tem um computador dentro dele. Hoje, é certo que televisores, telefones celulares, agendas eletrônicas, fornos de micro-ondas, filmadoras, aparelhos de DVD, impressoras a laser, alarmes antifurto, aparelhos de surdez, jogos eletrônicos e outros incontáveis dispositivos são todos controlados por computador. Os computadores que estão dentro desses aparelhos costumam ser otimizados para baixo preço e não para alto desempenho, o que provoca compromissos diferentes dos feitos para CPUs de tecnologia avançada que estudamos até aqui.

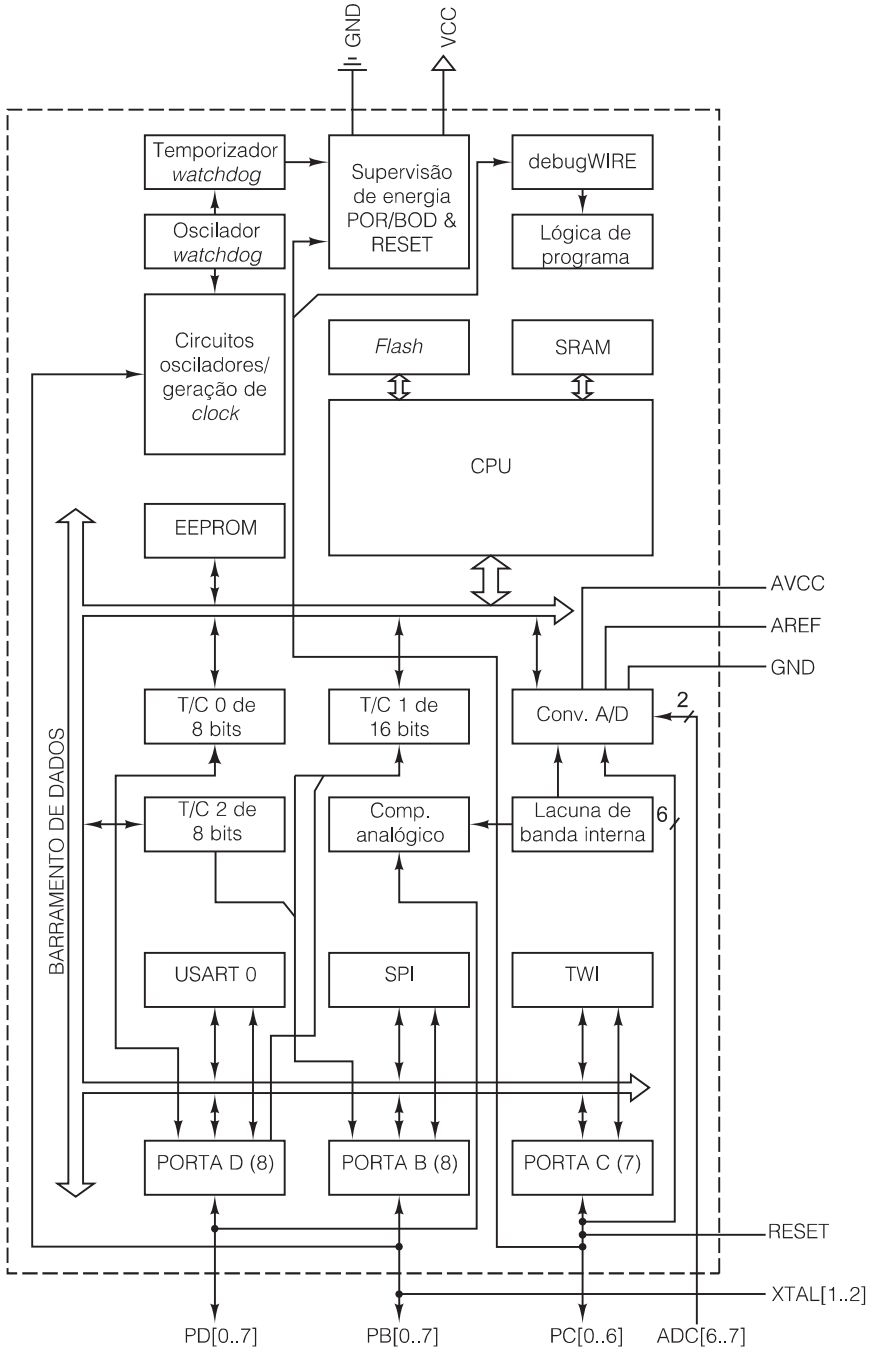
Como mencionamos no Capítulo 1, o Atmel ATmega168 provavelmente é o microcontrolador mais popular em uso hoje, em grande parte por causa de seu custo muito baixo (cerca de 1 dólar). Como veremos em breve, ele também é um chip versátil, portanto, fazer interface com ele é algo simples e barato. Agora, vamos examinar esse chip, cuja pinagem física é mostrada na Figura 3.49.

Figura 3.49 Pinagem física do ATmega168.



Como podemos ver na figura, o ATmega168 normalmente vem em um pacote padrão de 28 pinos, embora haja outros pacotes disponíveis. À primeira vista, você talvez tenha notado que a pinagem nesse chip é um pouco estranha em comparação com os dois projetos anteriores que examinamos. Em particular, esse chip não tem linhas de endereço e dados. Isso porque não foi projetado para ser conectado à memória, só a dispositivos. Toda a memória, SRAM e *flash*, está contida dentro do processador, evitando a necessidade de quaisquer pinos de endereço e dados, como mostra a Figura 3.50.

Figura 3.50 Arquitetura interna e pinagem lógica do ATmega168.



Em vez de pinos de endereço e dados, o ATmega168 tem 27 portas de E/S digitais, 8 linhas na porta B e D, e 7 linhas na porta C. Essas linhas de E/S digitais são projetadas para serem conectadas aos periféricos de E/S, e cada uma pode ser configurada internamente pelo software de partida para ser uma entrada ou uma saída. Por exemplo, quando usada em um forno de micro-ondas, uma linha de E/S digital seria uma entrada do sensor de "porta aberta". Outra linha de E/S digital seria uma saída usada para ligar e desligar o gerador do micro-ondas. O software no ATmega168 verificaria se a porta estava fechada antes de ligar o gerador do micro-ondas. Se a porta de repente for aberta, o software deverá cortar a energia. Na prática, as interconexões de hardware também estão sempre presentes.

Como opção, seis das entradas da porta C podem ser configuradas para serem E/S analógica. Pinos de E/S analógica podem ler o nível de tensão de uma entrada ou definir o nível de tensão de uma saída. Estendendo nosso exemplo de forno de micro-ondas, alguns aparelhos têm um sensor que permite ao usuário aquecer o alimento até determinada temperatura. O sensor de temperatura seria conectado a uma entrada de porta C, e o software poderia ler a tensão do sensor e depois convertê-la em uma temperatura usando uma função de tradução específica do sensor. Os pinos restantes no ATmega168 são a entrada de tensão (VCC), dois pinos de terra (GND) e dois pinos para configurar os circuitos de E/S analógica (AREF, AVCC).

A arquitetura interna do ATmega168, como a do OMAP4430, é um sistema-em-um-chip com uma rica matriz de dispositivos internos e memória. O ATmega168 vem com até 16 KB de memória *flash* interna, para armazenamento de informações não voláteis que mudam com pouca frequência, como instruções de programa. Ele também inclui até 1 KB de EEPROM, a memória não volátil que pode ser gravada pelo software. A EEPROM guarda dados de configuração do sistema. De novo, usando nosso exemplo de micro-ondas, a EEPROM armazenaria um bit indicando se o micro-ondas mostrará a hora em formato de 12 ou 24 horas. O ATmega168 também incorpora até 1 KB de SRAM interna, onde o software pode armazenar variáveis temporárias.

O processador interno roda o conjunto de instruções AVR, que é composto de 131 instruções, cada uma com 16 bits de extensão. O processador tem 8 bits, o que significa que opera sobre valores de dados de 8 bits, e internamente seus registradores possuem um tamanho de 8 bits. O conjunto de instruções incorpora instruções especiais que permitem ao processador de 8 bits operar de modo eficiente sobre tipos de dados maiores. Por exemplo, para realizar adições de 16 bits ou maiores, o processador fornece a instrução "*add-with-carry*" (somar com vai-um), que soma dois valores e mais o "vai-um" da adição anterior. Os outros componentes internos englobam o *clock* em tempo real e uma variedade de lógica de interface, incluindo suporte para enlaces seriais, enlaces PWM (*pulse-width-modulated* – modulado por largura de pulso), enlaces I2C (barramento Inter-IC) e controladores analógico e digital.

3.6 Exemplos de barramentos

Barramentos são a cola que mantém a integridade dos sistemas de computadores. Nesta seção, examinaremos minuciosamente alguns barramentos populares: o PCI e o USB (Universal Serial Bus – barramento serial universal). O PCI é o principal barramento de E/S usado hoje em dia nos PCs. Ele pode ter duas formas, o barramento PCI mais antigo, e o novo e muito mais rápido barramento PCI Express (PCIe). O Universal Serial Bus é um barramento de E/S cada vez mais popular para periféricos de baixa velocidade, como mouses e teclados. Uma segunda e terceira versões do barramento USB rodam com velocidades muito mais altas. Nas próximas seções, veremos esses barramentos um por vez, para ver como eles funcionam.

3.6.1 O barramento PCI

No IBM PC original, a maioria das aplicações era baseada em texto. De modo gradual, com a introdução do Windows, pouco a pouco começaram a ser usadas as interfaces gráficas de usuário. Nenhuma dessas aplicações exigia demais do barramento ISA. Contudo, com o passar do tempo, quando muitas aplicações, em especial jogos em multimídia, começaram a usar computadores para exibir vídeo em tela cheia e com movimento completo, a situação mudou radicalmente.

Vamos fazer um cálculo simples. Considere um vídeo colorido de 1.024 × 768 com 3 bytes/*pixel*. Um quadro contém 2,25 MB de dados. Para um movimento suave, são necessárias ao menos 30 telas por segundo para uma taxa de dados de 67,5 MB por segundo. Na verdade, é pior do que isso, pois para apresentar um vídeo a partir de um disco rígido, CD-ROM ou DVD, os dados devem passar do *drive* de disco para o barramento e ir até a memória. Então, para a apresentação, os dados devem novamente percorrer o barramento até o adaptador gráfico. Portanto, precisamos de uma largura de banda de barramento de 135 MB por segundo só para o vídeo, sem contar a largura de banda de que a CPU e outros dispositivos precisam.

Os estágios *Ex* (Execução) são onde as instruções são de fato executadas. Quase todas as instruções aritméticas, booleanas e de deslocamento utilizam as ULAs de inteiros e são concluídas em um ciclo. Cargas e armazenamentos utilizam dois ciclos (se estiverem presentes na *cache* L1), e multiplicações utilizam três ciclos. Os estágios *Ex* contêm várias unidades funcionais, que são:

1. ULA 1 de inteiros.
2. ULA 2 de inteiros.
3. Unidade de multiplicação.
4. ULA de ponto flutuante e vetor de SIMD (opcional com suporte a VFP e NEON).
5. Unidade de carga e armazenamento (*load/store*).

Instruções de desvio condicional também são processadas no primeiro estágio *Ex* e sua direção (desvio/sem desvio) é determinada. No caso de um erro de previsão, um sinal é enviado de volta ao estágio *Fe*1 e o *pipeline* é anulado.

Depois de concluir sua execução, as instruções entram no estágio *WB* (WriteBack), onde cada uma atualiza de imediato o arquivo de registrador físico. Depois, quando a instrução é a mais antiga em andamento, ela gravará o resultado do seu registrador no arquivo arquitetônico de registradores. Se houver uma interrupção, são esses valores, e não os dos registradores físicos, que se tornam visíveis. O ato de armazenar o registrador no arquivo arquitetônico é equivalente à retirada no Core i7. Além disso, no estágio *WB*, quaisquer instruções de armazenamento agora completam a escrita de seus resultados na *cache* de dados L1.

Essa descrição do Cortex A9 está longe de ser completa, mas deve dar uma ideia razoável de como ele funciona e de quais são as diferenças entre sua microarquitetura e a do Core i7.

4.6.3 A microarquitetura do microcontrolador ATmega168

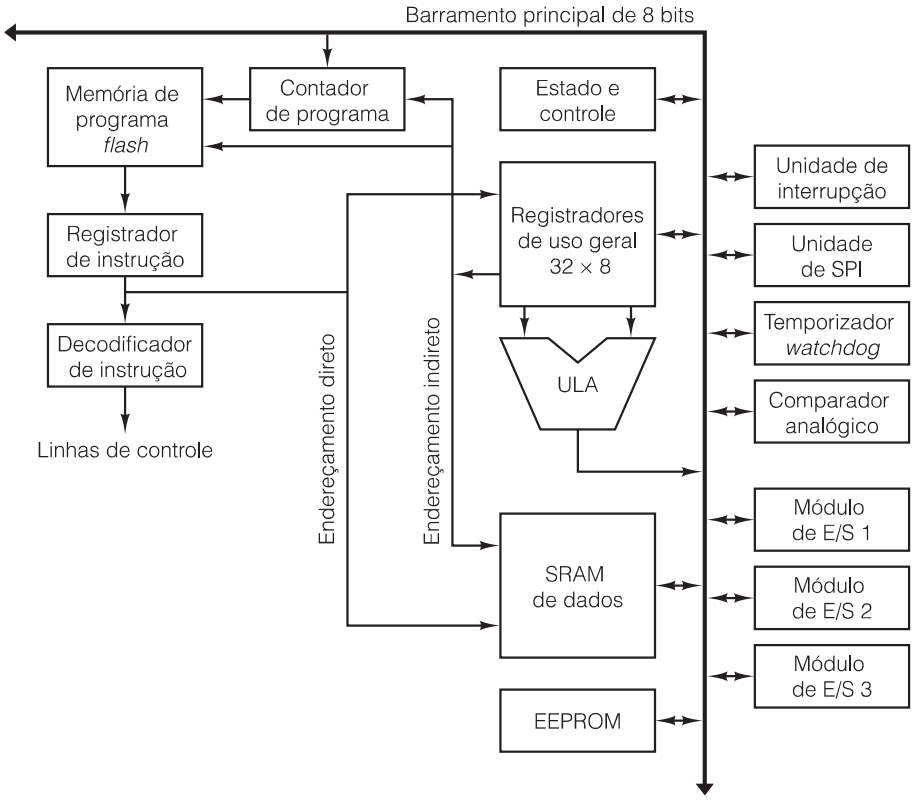
Nosso último exemplo de uma microarquitetura é a da Atmel ATmega168, mostrada na Figura 4.50. Essa microarquitetura é bem mais simples do que as do Core i7 e do OMAP4430. A razão para essa simplicidade é que o chip é muito pequeno e barato para atender ao mercado de projetos embutidos. Dessa forma, o objetivo principal era fazer um chip barato, não rápido. Barato e simples são bons amigos; barato e rápido, não.

O coração do ATmega168 é o barramento principal de 8 bits. Ligado a ele estão vários registradores e bits de estado, ULA, memória e dispositivos de E/S. Vamos descrevê-los brevemente agora. O arquivo de registradores contém 32 registradores de 8 bits, que são usados para armazenar valores temporários do programa. O registrador de **estado e controle** mantém os códigos de condição da última operação da ULA (ou seja, sinal, excesso, negativo, zero e vai-um), mais um bit que indica se uma interrupção está pendente. O **contador de programa** mantém o endereço da instrução em execução. Para realizar uma operação na ULA, primeiro os operandos são lidos do registrador e enviados à ULA. A saída da ULA pode ser escrita em qualquer um dos registradores passíveis de escrita por meio do barramento principal.

O ATmega168 tem diversas memórias para dados e instruções. A SRAM de dados tem 1 KB, muito grande para ser totalmente endereçada com um endereço de 8 bits no barramento principal. Assim, a arquitetura AVR permite que os endereços sejam construídos com um par sequencial de registradores de 8 bits, produzindo assim um endereço de 16 bits que admite até 64 KB de memória de dados. A EEPROM oferece até 1 KB de armazenamento não volátil, onde os programas podem escrever variáveis que precisam sobreviver a uma falta de energia.

Existe um mecanismo semelhante para endereçar a memória do programa, mas 64 KB de código é muito pouco, até mesmo para sistemas embutidos, de baixo custo. Para permitir que mais memória de instruções seja endereçada, a arquitetura AVR define três registradores de página de RAM (RAMPX, RAMPY e RAMPZ), cada um com 8 bits de largura. O registrador de página de RAM é concatenado com um par de registradores de 16 bits para produzir um endereço de programa de 24 bits, permitindo assim 16 MB de espaço de endereço de instruções.

Figura 4.50 Microarquitetura do ATmega168.



Para e pense nisso por um instante. 64 KB de código é muito pouco para um microcontrolador que poderia controlar um brinquedo ou um pequeno aparelho. Em 1964, a IBM lançou o System 360 Model 30, que tinha 64 KB de memória total (sem truques para aumentá-la). Ele era vendido por US$ 250 mil, que é cerca de US$ 2 milhões em dólares de hoje. O ATmega168 custa cerca de US$ 1, ou menos se você comprar em quantidade. Se você verificar, digamos, o custo de venda de um Boeing, verá que os preços de aeronaves não caíram por um fator de 250.000 nos últimos 50 anos ou mais. E nem os valores de carros ou televisores, ou qualquer outra coisa, exceto computadores.

Além disso, o ATmega168 tem um controlador de interrupção no chip, interface de porta serial (SPI) e temporizadores, que são essenciais para aplicações de tempo real. Há também três portas de E/S digitais de 8 bits, que lhe permitem controlar até 24 botões externos, luzes, sensores, acionadores e assim por diante. É a presença dos temporizadores e portas de E/S, mais do que qualquer outra coisa, que possibilita o uso do ATmega168 para aplicações embutidas sem quaisquer chips adicionais.

O ATmega168 é um processador síncrono, com a maior parte das instruções usando apenas um ciclo de *clock*, embora algumas usem mais. O processador é paralelo, de modo que, enquanto uma instrução está sendo buscada, a anterior está sendo executada. Entretanto, o *pipeline* tem apenas dois estágios, busca e execução. Para executar instruções em um ciclo, o ciclo de *clock* deve acomodar a leitura do registrador do arquivo de registradores, seguida pela execução da instrução na ULA, seguida pela escrita do registrador de volta ao arquivo de registradores. Como todas essas operações ocorrem em um ciclo de *clock*, não é preciso de lógica de contorno (*bypass*) ou detecção de protelação (*stall*). As instruções do programa são executadas em ordem, em um ciclo, e sem sobreposição com outras instruções.

Embora pudéssemos entrar em mais detalhes sobre o ATmega168, a descrição que demos e a Figura 4.50 oferecem uma ideia básica. O ATmega168 tem um único barramento principal (para reduzir a área do chip),

um conjunto heterogêneo de registradores e uma série de memórias e dispositivos de E/S pendurados no barramento principal. A cada ciclo do caminho de dados, dois operandos são lidos do arquivo de registradores e passam pela ULA, com os resultados enviados de volta a um registrador, assim como nos computadores mais modernos.

4.7 Comparação entre i7, OMAP4430 e ATmega168

Nossos três exemplos são muito diferentes, porém, ainda assim exibem certa dose de características em comum. O Core i7 tem um conjunto de instruções CISC antigo que os engenheiros da Intel adorariam jogar na Baía de São Francisco, caso isso não violasse as leis antipoluição das águas da Califórnia. O OMAP4430 é um projeto RISC puro, com um conjunto de instruções enxuto e esperto. O ATmega168 é um processador simples de 8 bits para aplicações embutidas. Ainda assim, o coração de cada um deles é um conjunto de registradores e uma ou mais ULAs que efetuam operações aritméticas e booleanas simples em operandos de registradores.

A despeito de suas óbvias diferenças externas, o Core i7 e o OMAP4430 têm unidades de execução bastante semelhantes. Ambas as unidades de execução aceitam micro-operações que contêm um *opcode*, dois registradores de origem e um registrador de destino. Ambos podem executar uma micro-operação em um ciclo. Ambos têm alto grau de *pipelining*, previsão de desvio e *caches* de instruções (I) e de dados (D) divididas.

Essa semelhança interna não é um acidente ou nem mesmo causada pela eterna rotatividade de empregos dos engenheiros do Vale do Silício. Como vimos em nossos exemplos de Mic-3 e Mic-4, é fácil e natural construir um caminho de dados com *pipeline* que pega dois registradores de origem, passa-os por uma ULA e armazena os resultados em um registrador. A Figura 4.34 mostra esse *pipeline* graficamente. Com a tecnologia atual, esse é o projeto mais eficaz.

A principal diferença entre as CPUs Core i7 e OMAP4430 é o modo como elas vão de seu conjunto de instrução ISA até a unidade de execução. O Core i7 tem de fragmentar suas instruções CISC para colocá-las no formato de três registradores de que a unidade de execução necessita. É isso que faz o terminal frontal na Figura 4.47 – desmembra instruções grandes em micro-operações caprichadas e jeitosas. O OMAP4430 não tem de fazer nada porque suas instruções nativas já são micro-operações caprichadas e jeitosas. É por isso que grande parte das novas ISAs são do tipo RISC – para oferecer melhor compatibilidade entre o conjunto de instruções ISA e o mecanismo interno de execução.

É instrutivo comparar nosso projeto final, a Mic-4, com esses dois exemplos do mundo real. A Mic-4 é muito parecida com o Core i7. A tarefa de ambos é interpretar um conjunto de instrução ISA que não é RISC. Ambos fazem isso desmembrando as instruções ISA em micro-operações com um *opcode*, dois registradores de origem e um de destino. Em ambos os casos, as micro-operações são depositadas em uma fila para execução mais tarde. A política estrita do projeto da Mic-4 prevê emissão, execução, retirada em ordem, ao passo que o Core i7 tem uma política de emissão em ordem, execução fora de ordem, retirada em ordem.

Na realidade, Mic-4 e OMAP4430 não podem ser comparados, porque o conjunto de instruções ISA do OMAP4430 é composto de instruções RISC (isto é, micro-operações de três registradores). Essas instruções não têm de ser desmembradas e podem ser executadas como se apresentam, cada uma em um único ciclo de caminho de dados.

Em comparação com Core i7 e OMAP4430, o ATmega168 é realmente uma máquina simples. Tende mais para RISC do que para CISC porque grande parte de suas instruções simples pode ser executada em um ciclo de *clock* e não precisa ser desmembrada. Ele não tem *pipelining*, nem *cache*, e tem emissão, execução e retirada em ordem. Em sua simplicidade, é muito mais semelhante à Mic-1.

4.8 Resumo

O coração de todo computador é o caminho de dados. Ele contém alguns registradores, um, dois ou três barramentos e uma ou mais unidades funcionais, como ULAs e deslocadores. O laço de execução principal consiste