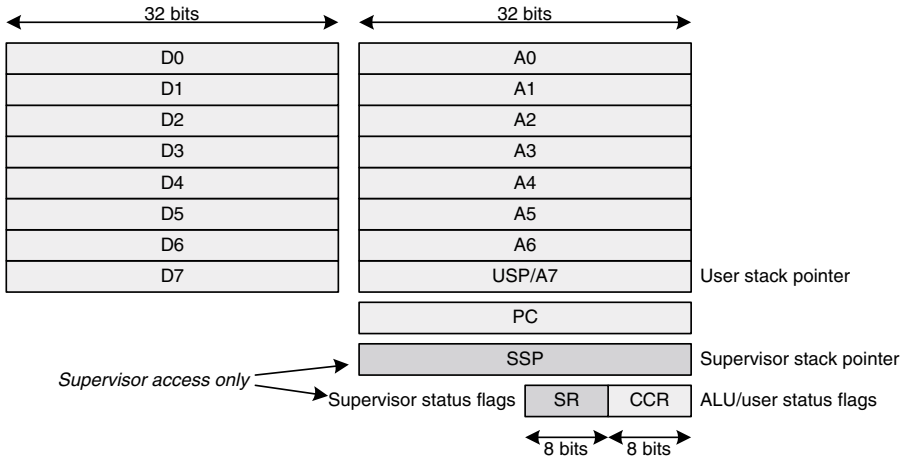

FIGURE 6.11 68000 register set.

ALU operations. All 16 registers can be used as index registers. While operating in user mode, it is illegal to access the SSP or the supervisor portion of the status register, SR. Such instructions will cause an exception, whereby a particular interrupt is asserted, which causes the 68000 to enter supervisor mode to handle the fault. (*Exception* and *interrupt* are often used synonymously in computer contexts.) Very often, the OS kernel will terminate an application that causes an exception to be generated. The registers shown above are present in all 68000 family members and, as such, are software is compatible with subsequent 68xxx microprocessors. Newer microprocessors contain additional registers that provide more advanced privilege levels and memory management. While the 68000 architecture fundamentally supports a 4-GB (32-bit) address space, early devices were limited in terms of how much physical memory could actually be addressed as a result of pin limitations in the packaging. The original 68000 was housed in a 64-pin DIP, leaving only 24 address bits usable, for a total usable memory space of 16 MB. When Motorola introduced the 68020, the first fully 32-bit 68000 microprocessor, all 32 address bits were made available. The 68000 devices are big-endian, so the MSB is stored in the lowest address of a multibyte word.

The 68000 supports a 16-MB address space, but only 23 address bits, A[23:1], are actually brought out of the chip as signal pins. A[0] is omitted and is unnecessary, because it would specify whether an even (A[0] = 0) or odd (A[0] = 1) byte is being accessed; and, because the bus is 16 bits wide, both even and odd bytes can be accessed simultaneously. However, provisions are made for byte-wide accesses in situations where the 68000 is connected to legacy eight-bit peripherals or memories. Two data strobes, upper (UDS*) and lower (LDS*), indicate which bytes are being accessed during any given bus cycle. These strobes are generated by the 68000 according to the state of the internal A0 bit and information on the size of the requested transaction. Bus transactions are triggered by the assertion of address strobe (AS*), the appropriate data strobes, and R/W* as shown in Fig. 6.12. Prior to AS*, the 68000 asserts the desired address and a three-bit function code bus, FC[2:0]. The function code bus indicates which mode the processor is in and whether the transaction is a program or data access. This information can be used by external logic to qualify transactions to certain sensitive memory spaces that may be off limits to user programs. When read data is ready, the external bus interface logic asserts data transfer acknowledge (DTACK*) to inform the microprocessor that the transaction is complete. As shown, the 68000 bus can be operated in a fully asynchronous manner. When operated asynchronously, DTACK* is removed after the strobes are

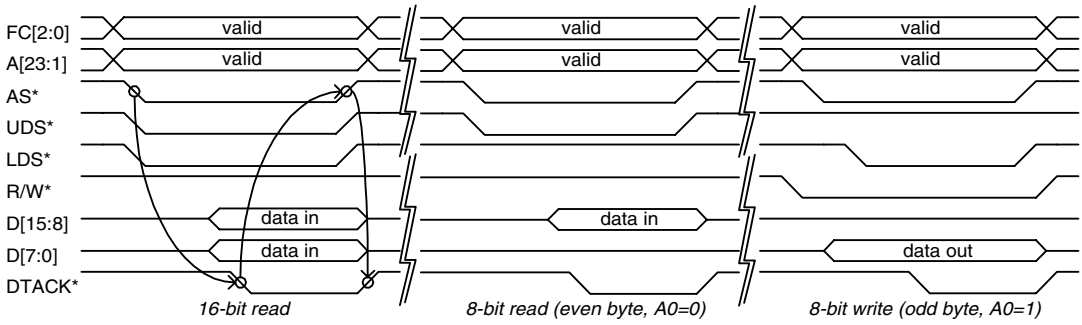

FIGURE 6.12 68000 asynchronous bus timing.

removed, ensuring that the 68000 detected the assertion of DTACK*. If DTACK* is removed prior to the strobes, there is a chance of marginal timing where the 68000 may not properly detect the acknowledge, and it may wait forever for an acknowledge that has now passed. Writes are very similar to reads, with the obvious difference that R/W* is brought low, and data is driven by the 68000. Another difference is that the data strobe assertion lags that of AS*.

Advanced microprocessors such as the 68000 are designed to recognize fault conditions wherein the requested bus transaction cannot be completed. A bus fault can be caused by a variety of problems, including unauthorized access (e.g., user mode tries to write to a protected supervisor data space) or an access to a section of memory that is not filled by a memory or peripheral device. Software should never access areas of memory that are off limits, because the results are unpredictable. Therefore, rather than simply issuing a false DTACK* and continuing with normal operation, the 68000 contains a bus error signal (BERR*) that behaves like DTACK* but triggers an exception rather than continuing normal execution. It is the responsibility of external logic to manage the DTACK* and BERR* signals according to the specific configuration and requirements of the particular system.

Operating the 68000 bus in an asynchronous manner is easy, but it reduces its bandwidth, because delays must be built into the acknowledge process to guarantee that both the 68000 and the interface logic maintain synchronization. Figure 6.12 shows read data being asserted prior to DTACK* and an arbitrary delay between the release of AS* and that of DTACK*. The data delay is necessary to guarantee that the 68000 will see valid data when it detects a valid acknowledge. The second delay is necessary to ensure that the 68000 completes the transaction, as noted previously. These delays can be eliminated if the bus is operated synchronously by distributing the microprocessor clock to the interface logic and guaranteeing that various setup and hold timing requirements are met as specified by Motorola. In such a configuration, it is known from Motorola's data sheet that the 68000 looks for DTACK* each clock cycle, starting at a fixed time after asserting the strobes, and then samples the read-data one cycle after detecting DTACK* being active. Because synchronous timing rules are obeyed, it is guaranteed that the 68000 properly detects DTACK* and, therefore, DTACK* can be removed without having to wait for the removal of the strobes. 68000 synchronous bus timing is shown in Fig. 6.13, where each transaction lasts a minimum of four clock cycles. A four-cycle transaction is a zero wait state access. Wait states can be added by simply delaying the assertion of DTACK* to the next cycle. However, to maintain proper timing, DTACK* (and BERR* and read-data) must always obey proper setup and hold requirements. As shown in the timing diagram, each signal transition, or edge, is time-bounded relative to a clock edge.

Read timing allows a single clock cycle between data strobe assertion and the return of DTACK* for a zero wait-state transaction. However, zero wait-state writes require DTACK* assertion at