Memory Management: Simple Systems

Bibliography

- Understanding Operating Systems Seventh Edition by Ann McIver, McHoes Ida, M. Flynn; Chapter 2

Learning Objectives

After completing this chapter, you should be able to describe:

- The basic functionality of the four memory allocation schemes presented in this chapter: single user, fixed partitions, dynamic partitions, and relocatable dynamic partitions
- Best-fit memory allocation as well as first-fit memory allocation
- How a memory list keeps track of available memory

Learning Objectives (cont'd.)

- The importance of memory deallocation
- The importance of the bounds register in memory allocation schemes
- The role of compaction and how it can improve memory allocation efficiency

Introduction

- Main memory management is critical
- Entire system performance is dependent on two items:
  - Amount of memory available
  - Optimization of memory during job processing
    - **Memory optimization** is a range of techniques related to improving computer memory, such as identifying memory leaks and corruption, to optimize memory usage and increase performance and application usability
    - Memory leak is a type of resource leak that occurs when a computer program incorrectly manages memory allocations in a way that memory which is no longer needed is not released. A memory leak may also happen when an object is stored in memory but cannot be accessed by the running code

Introduction (cont'd.)

- This chapter introduces:
  - Role of main memory (RAM)
  - Four types: memory allocation schemes
    - Single-user systems
    - Fixed partitions
    - Dynamic partitions
    - Relocatable dynamic partitions

Single-User Contiguous Scheme

- Entire program: loaded into memory
- Contiguous memory space: allocated as needed
- Jobs: processed sequentially
- Memory Manager: performs minimal work
    1. Evaluates incoming process size: loads if small enough to fit; otherwise, rejects and evaluates next incoming process
    2. Monitors occupied memory space; when process ends, makes entire memory space available and returns to Step 1

Single-User Contiguous Scheme (cont'd.)

- Disadvantages
  - Multiprogramming or networking not supported
  - Not cost effective: unsuitable for business when introduced in late 1940s and early 1950s

Memory Allocation Schemes

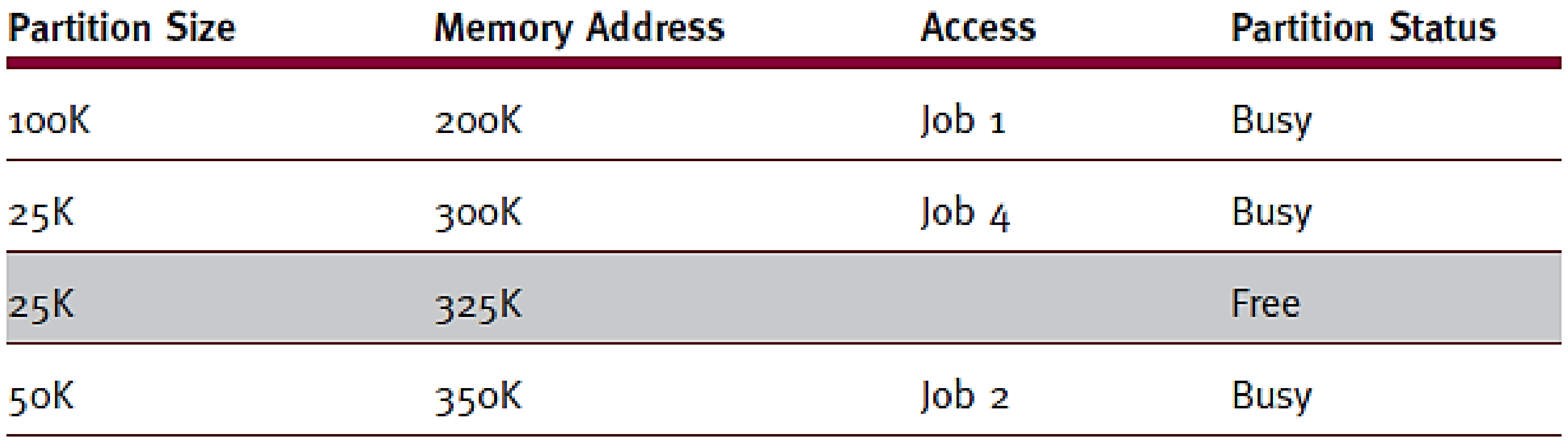
- Single-user systems
- Fixed partitions
- Dynamic partitions
- Relocatable dynamic partitions

Fixed Partitions

- Permits multiprogramming
- Main memory: partitioned
  - Each partition: one job
  - Static: reconfiguration requires system shut down
- Responsibilities
  - Protecting each job’s memory space
  - Matching job size with partition size

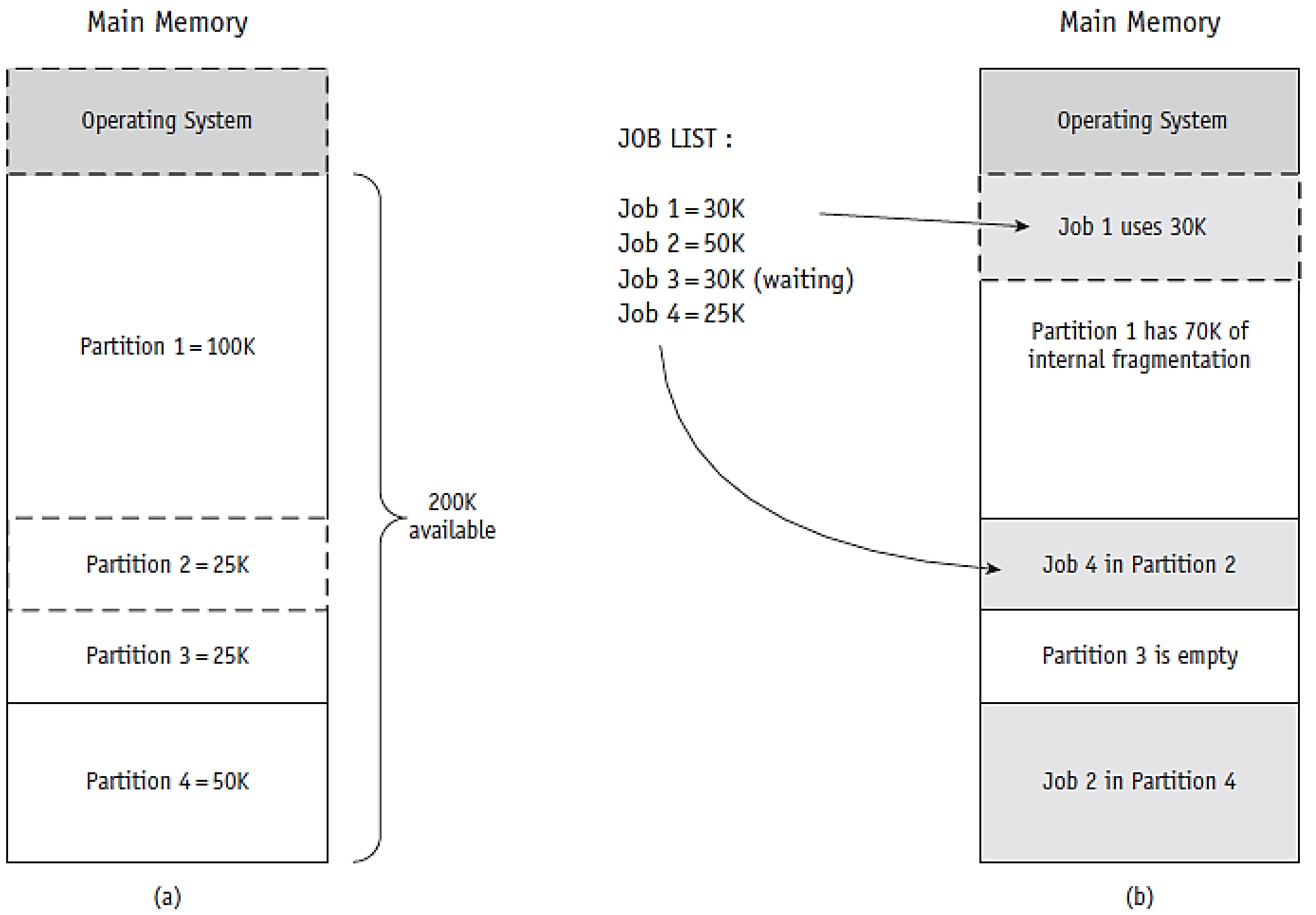
Fixed Partitions (cont'd.)

- Memory Manager: allocates memory space to jobs
  - Information: stored in a table



Partition Size	Memory Address	Access	Partition Status
100K	200K	Job 1	Busy
25K	300K	Job 4	Busy
25K	325K		Free
50K	350K	Job 2	Busy

(table 2.1)
A simplified fixed-partition memory table with the free partition shaded.
© Cengage Learning 2014



(figure 2.2)
As the jobs listed in Table 2.1 are loaded into the four fixed partitions,
Job 3 must wait even though Partition 1 has 70K of available memory. Jobs are allocated space on the basis of “first available partition of required size.”
© *Cengage Learning 2014*

Fixed Partitions (cont'd.)

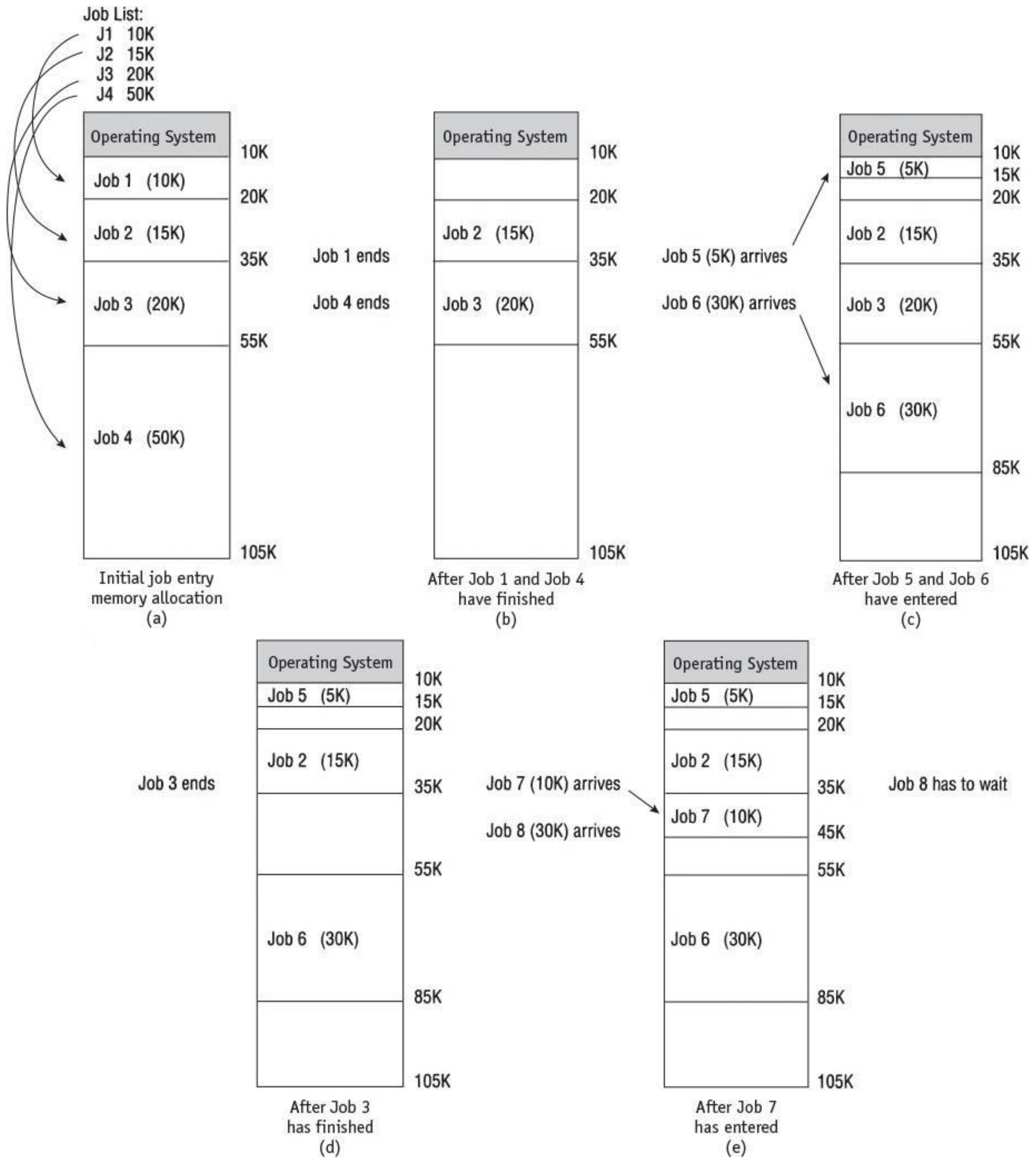
- Characteristics
  - Requires contiguous loading of entire program
  - Job allocation method
    - First available partition with required size
  - To work well:
    - All jobs have similar size and memory size known ahead of time
  - Arbitrary partition size leads to undesired results
    - Partition too small
      - Large jobs have longer turnaround time
    - Partition too large
      - Memory waste: internal fragmentation

Memory Allocation Schemes

- Single-user systems
- Fixed partitions
- Dynamic partitions
- Relocatable dynamic partitions

Dynamic Partitions

- Main memory: partitioned
  - Jobs: given requested memory when *loaded*
  - One contiguous partition per job
- Job allocation method
  - First come, first serve
  - Memory waste: comparatively small within partitions
- Disadvantages
  - Full memory utilization: only when first jobs loaded
  - Subsequent allocation: memory waste
    - External fragmentation: fragments between blocks



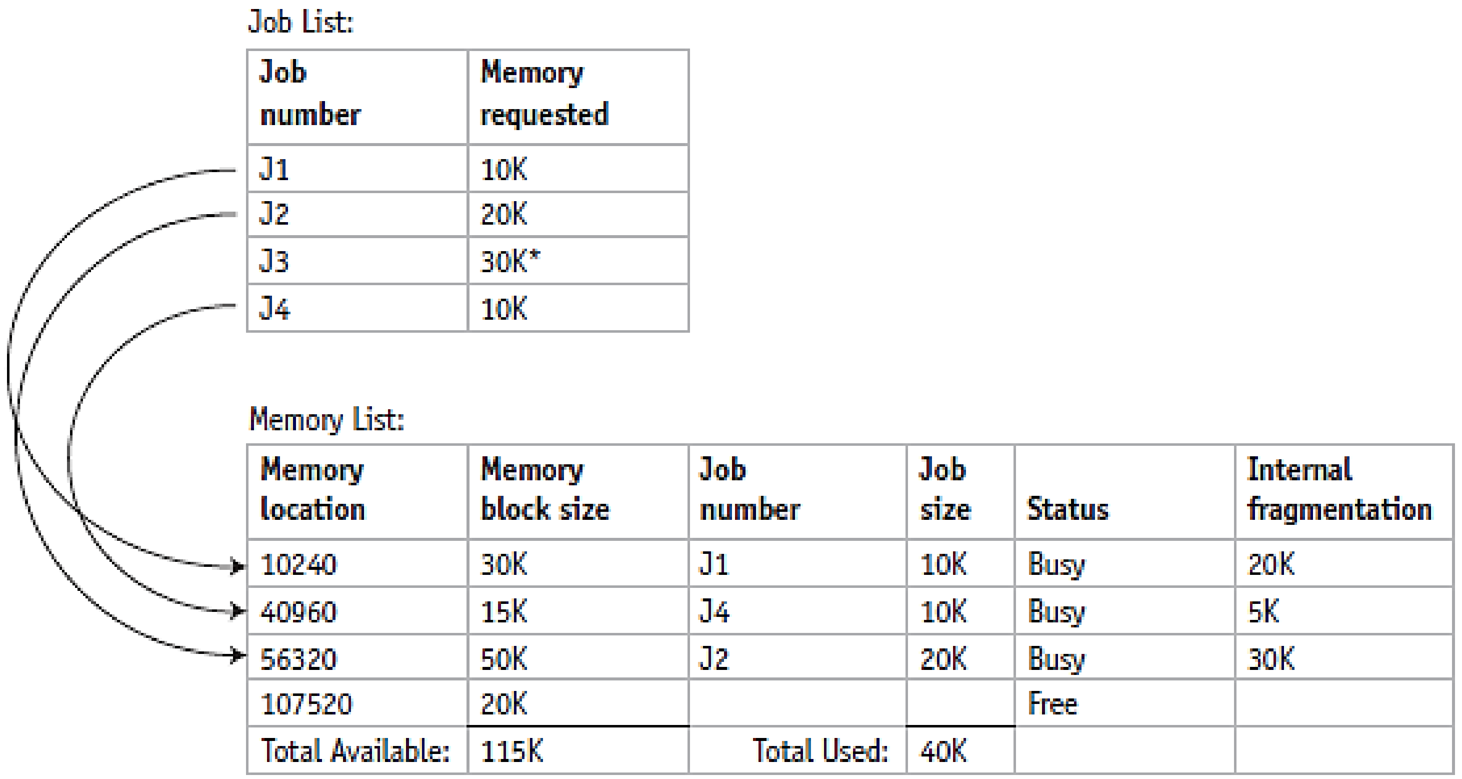
(figure 2.3)
Main memory use during dynamic partition allocation. Five snapshots (a-e) of main memory as eight jobs are submitted for processing and allocated space on the basis of "first come, first served." Job 8 has to wait (e) even though there's enough free memory between partitions to accommodate it.
© Cengage Learning 2014

Best-Fit and First-Fit Allocation

- Two methods for free space allocation
  - First-fit memory allocation
    - Memory Manager: free/busy lists organized by memory locations (low- to high-order memory)
    - Job: assigned first partition large enough
    - Fast allocation
  - Best-fit memory allocation
    - Memory Manager: free/busy lists ordered by size (smallest to largest)
    - Job: assigned smallest partition large enough
    - Least wasted space; internal fragmentation reduced

Best-Fit and First-Fit Allocation (cont'd.)

- Fixed and dynamic memory allocation schemes: use both methods
- First-fit memory allocation
  - Advantage: faster allocation
  - Disadvantage: memory waste
- Best-fit memory allocation
  - Advantage: best use of memory space
  - Disadvantage: slower allocation

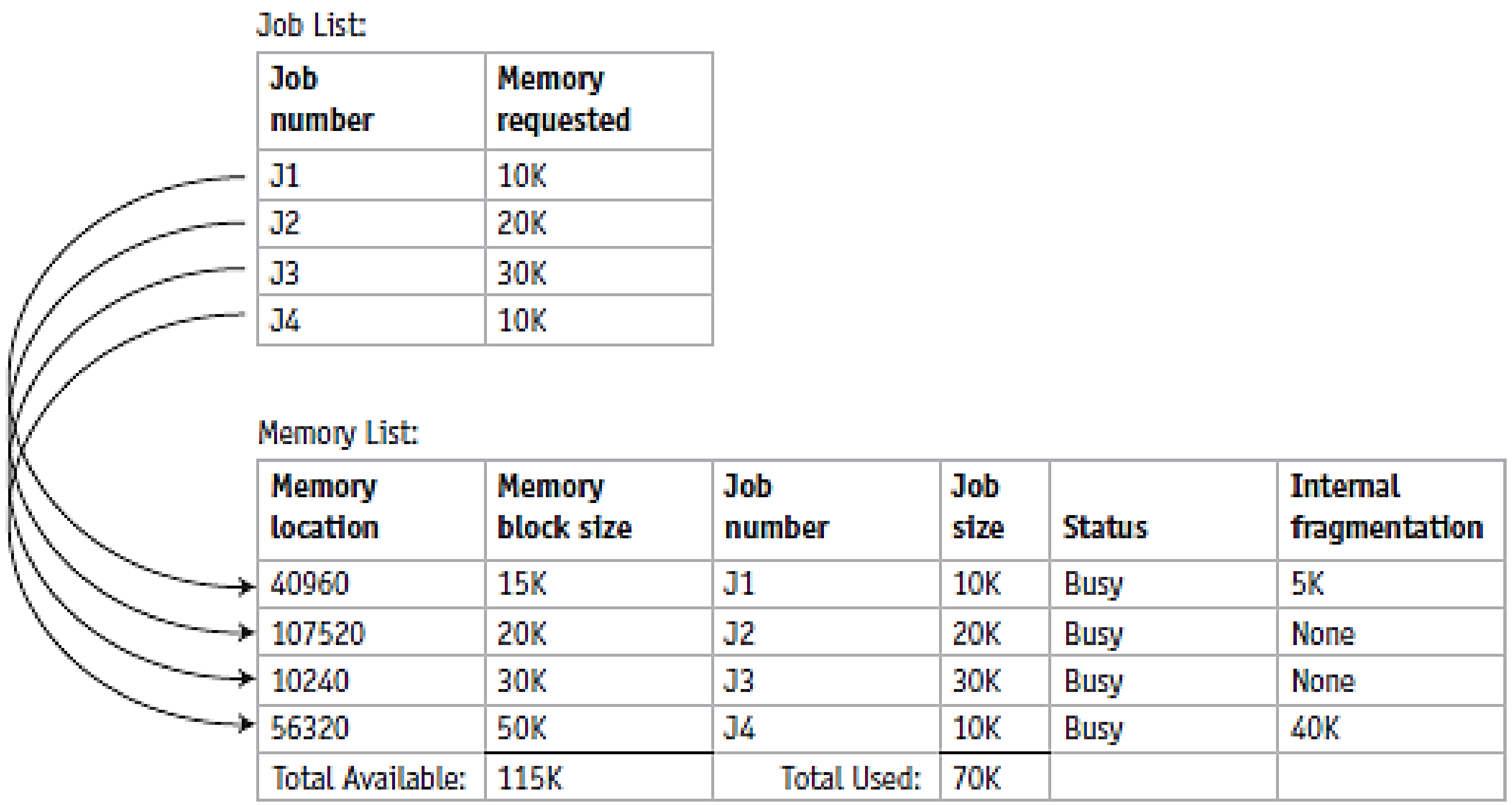

Job List:

Job number	Memory requested
J1	10K
J2	20K
J3	30K*
J4	10K

Memory List:

Memory location	Memory block size	Job number	Job size	Status	Internal fragmentation
10240	30K	J1	10K	Busy	20K
40960	15K	J4	10K	Busy	5K
56320	50K	J2	20K	Busy	30K
107520	20K			Free	
Total Available:	115K	Total Used:	40K		

(figure 2.5)
Using a first-fit scheme, Job 1 claims the first available space. Job 2 then claims the first partition large enough to accommodate it, but by doing so it takes the last block large enough to accommodate Job 3. Therefore, Job 3 (indicated by the asterisk) must wait until a large block becomes available, even though there's 75K of unused memory space (internal fragmentation). Notice that the memory list is ordered according to memory location.
© Cengage Learning 2014



Job List:

Job number	Memory requested
J1	10K
J2	20K
J3	30K
J4	10K

Memory List:

Memory location	Memory block size	Job number	Job size	Status	Internal fragmentation
40960	15K	J1	10K	Busy	5K
107520	20K	J2	20K	Busy	None
10240	30K	J3	30K	Busy	None
56320	50K	J4	10K	Busy	40K
Total Available:	115K	Total Used:	70K		

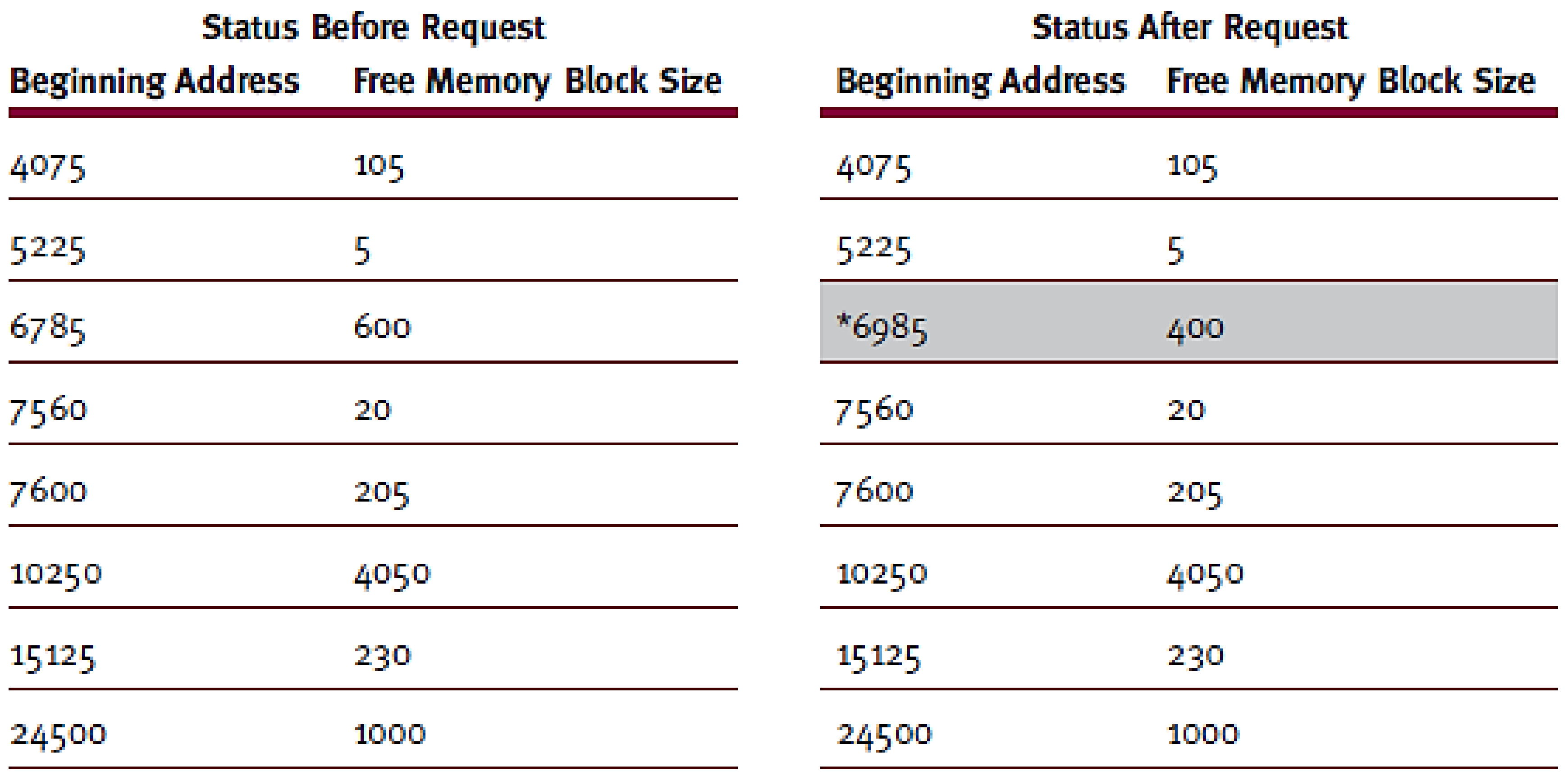
(figure 2.6)
Best-fit free scheme. Job 1 is allocated to the closest-fitting free partition, as are Job 2 and Job 3. Job 4 is allocated to the only available partition although it isn't the best-fitting one. In this scheme, all four jobs are served without waiting. Notice that the memory list is ordered according to memory size. This scheme uses memory more efficiently but it's slower to implement.
© Cengage Learning 2014

Best-Fit and First-Fit Allocation (cont'd.)

- First-fit algorithm
  - Memory Manager: keeps two lists
    - One for free memory
    - One for busy memory blocks
  - Loop: compares job size to each memory block size
    - Until large enough block found: fits the job
  - Job: stored into that memory block
  - Memory Manager: moves out of the loop
    - Fetches next job: entry queue

Best-Fit and First-Fit Allocation (cont'd.)

- First-fit algorithm (cont'd.)
  - If entire list searched: no memory block large enough
    - Job: placed into waiting queue
    - Memory Manager: fetches next job
  - Process repeats

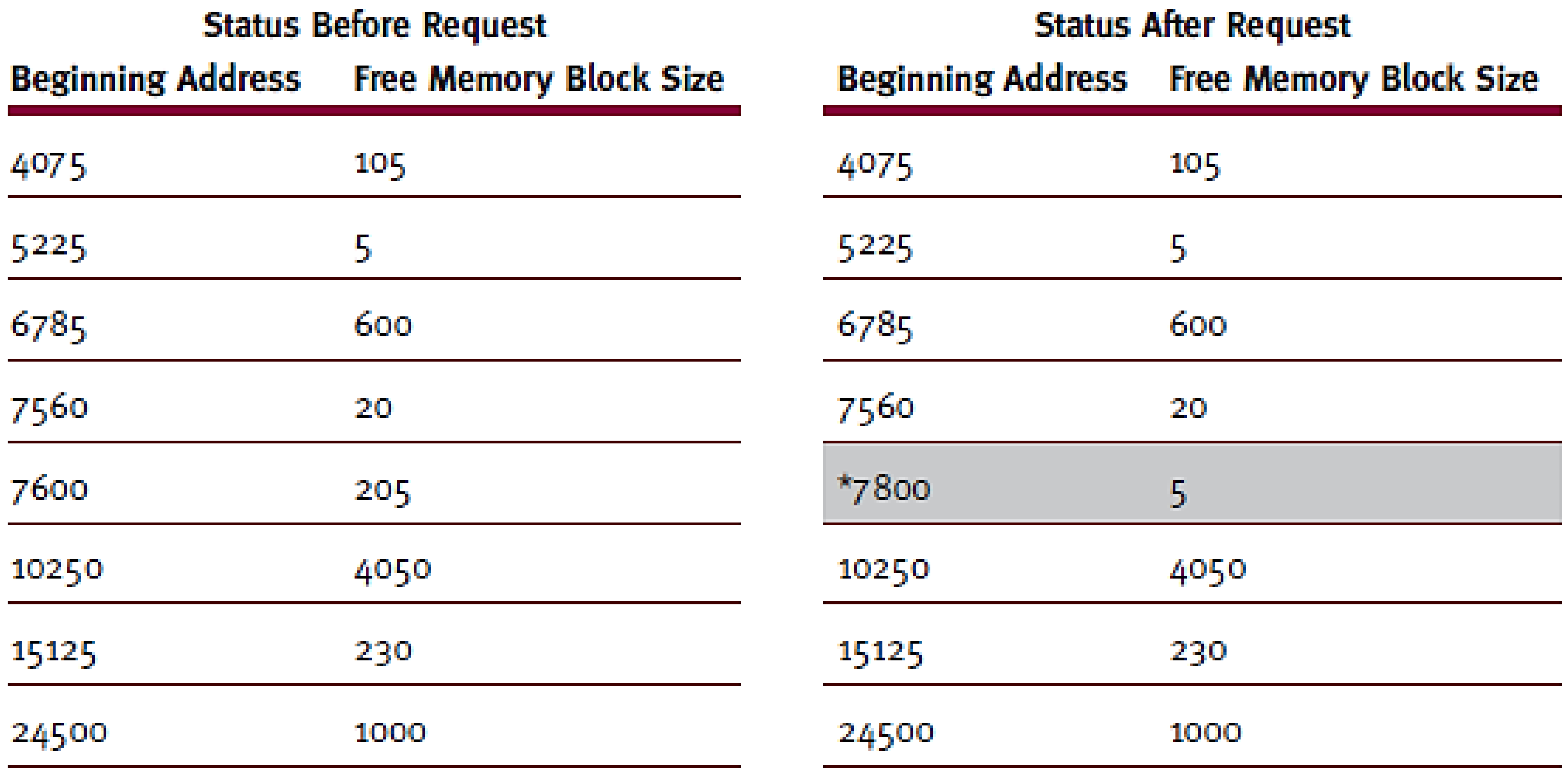


Status Before Request		Status After Request	
Beginning Address	Free Memory Block Size	Beginning Address	Free Memory Block Size
4075	105	4075	105
5225	5	5225	5
6785	600	*6985	400
7560	20	7560	20
7600	205	7600	205
10250	4050	10250	4050
15125	230	15125	230
24500	1000	24500	1000

(table 2.2)
These two snapshots of memory show the status of each memory block before and after 200 spaces are allocated at address 6785, using the first-fit algorithm. (Note: All values are in decimal notation unless otherwise indicated.)
© Cengage Learning 2014

Best-Fit Versus First-Fit Allocation (cont'd.)

- Best-fit algorithm
  - Goal: find smallest memory block where job fits
  - Entire table searched before allocation



Status Before Request		Status After Request	
Beginning Address	Free Memory Block Size	Beginning Address	Free Memory Block Size
4075	105	4075	105
5225	5	5225	5
6785	600	6785	600
7560	20	7560	20
7600	205	*7800	5
10250	4050	10250	4050
15125	230	15125	230
24500	1000	24500	1000

(table 2.3)
These two snapshots of memory show the status of each memory block before and after 200 spaces are allocated at address 7600, using the best-fit algorithm.
© Cengage Learning 2014

Best-Fit Versus First-Fit Allocation (cont'd.)

- Hypothetical allocation schemes
  - Next-fit: starts searching from last allocated block for next available block
  - Worst-fit: allocates largest free available block
    - Opposite of best-fit
    - Good way to explore theory of memory allocation
    - Not best choice for an actual system

Memory Allocation Schemes

- Single-user systems
- Fixed partitions
- Dynamic partitions
- Relocatable dynamic partitions
  - (Before discussing this scheme, we must understand Memory Deallocation)

Deallocation

- Deallocation: releasing allocated memory space
- For fixed-partition system:
  - Straightforward process
  - Memory Manager: resets the job’s memory block status to “free” upon job completion
  - Any code may be used: e.g., 0 = free and 1 = busy

Deallocation (cont'd.)

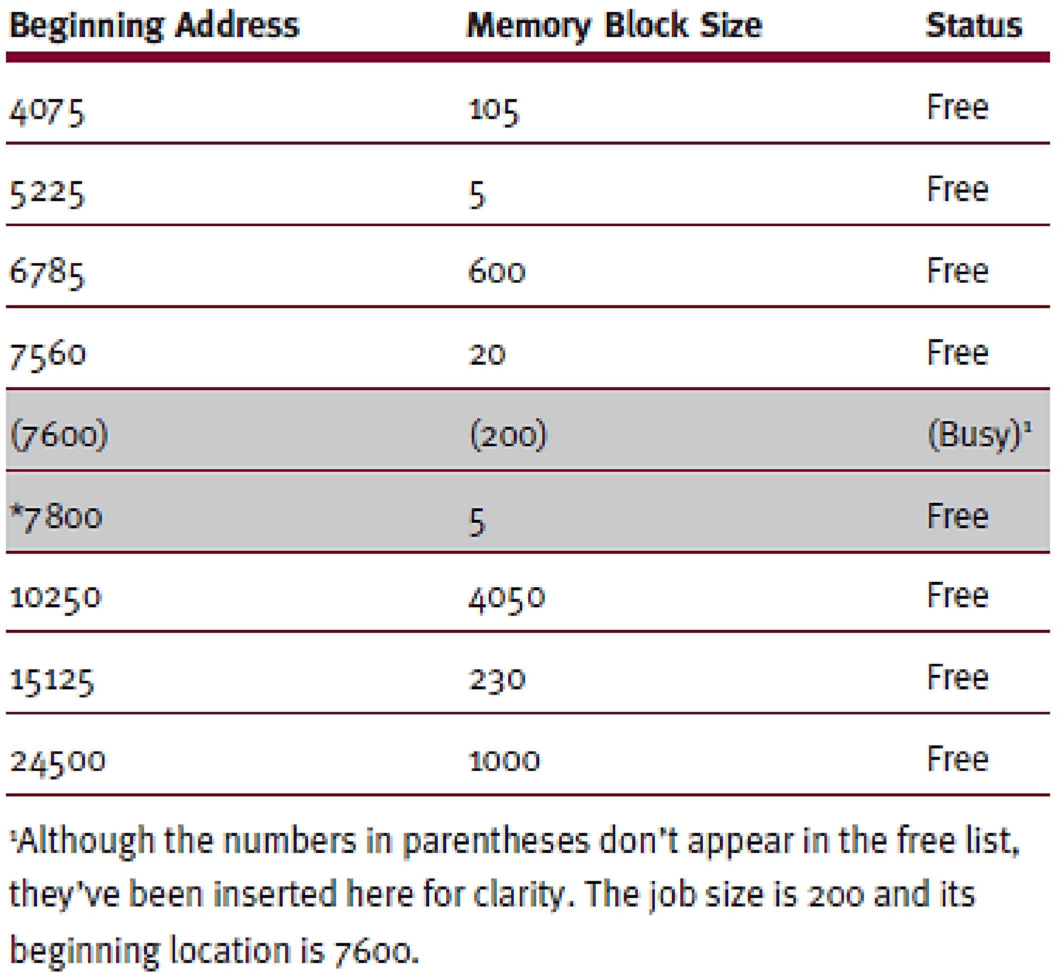
- For dynamic-partition system:
  - Algorithm: tries to combine free areas of memory
  - More complex
- Three dynamic partition system cases: depending on position of block to be deallocated
  - Case 1: adjacent to another free block
  - Case 2: between two free blocks
  - Case 3: isolated from other free blocks

Case 1: Joining Two Free Blocks

- Adjacent blocks
- List changes: reflect starting address of the new free block
  - Example: 7600 - address of the first instruction of the job that just released this block
- Memory block size changes: shows new size for the new free space
  - Combined total: two free partitions
  - Example: (200 + 5)

Case 1: Joining Two Free Blocks (cont'd.)

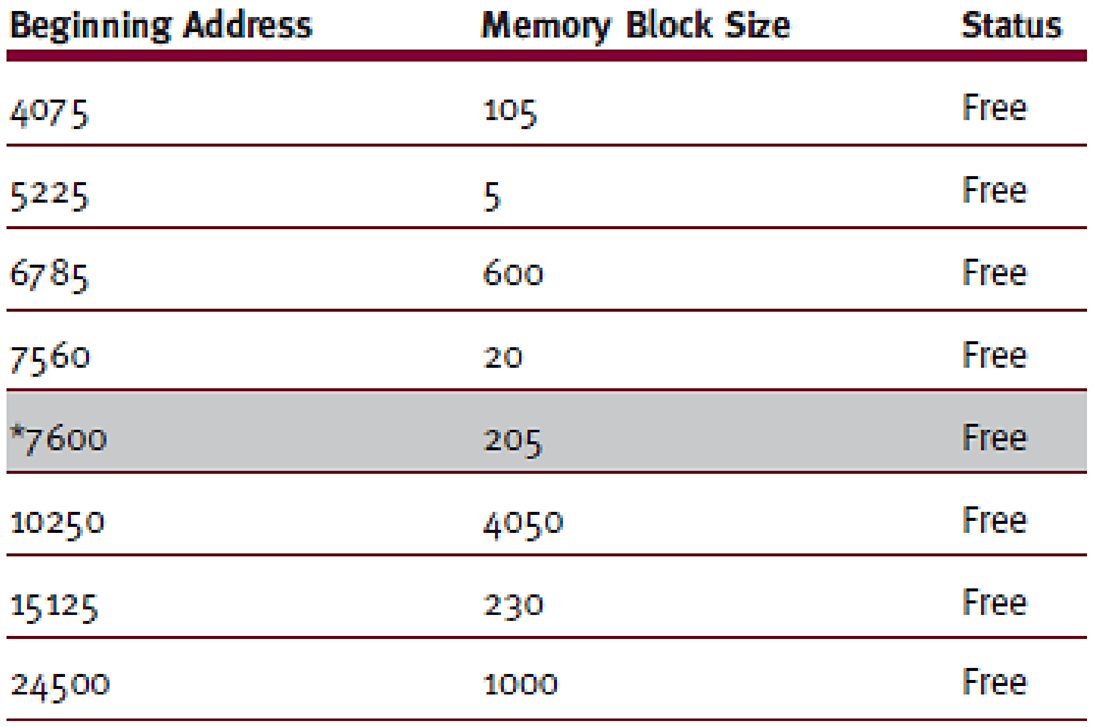
(table 2.4)
This is the original free list before deallocation for Case 1. The asterisk indicates the free memory block (of size 5) that’s adjacent to the soon-to-be-free memory block (of size 200) that’s shaded.
© *Cengage Learning 2014*



Beginning Address	Memory Block Size	Status
4075	105	Free
5225	5	Free
6785	600	Free
7560	20	Free
(7600)	(200)	(Busy)[1]
*7800	5	Free
10250	4050	Free
15125	230	Free
24500	1000	Free

[1]Although the numbers in parentheses don't appear in the free list, they've been inserted here for clarity. The job size is 200 and its beginning location is 7600.

Case 1: Joining Two Free Blocks (cont'd.)



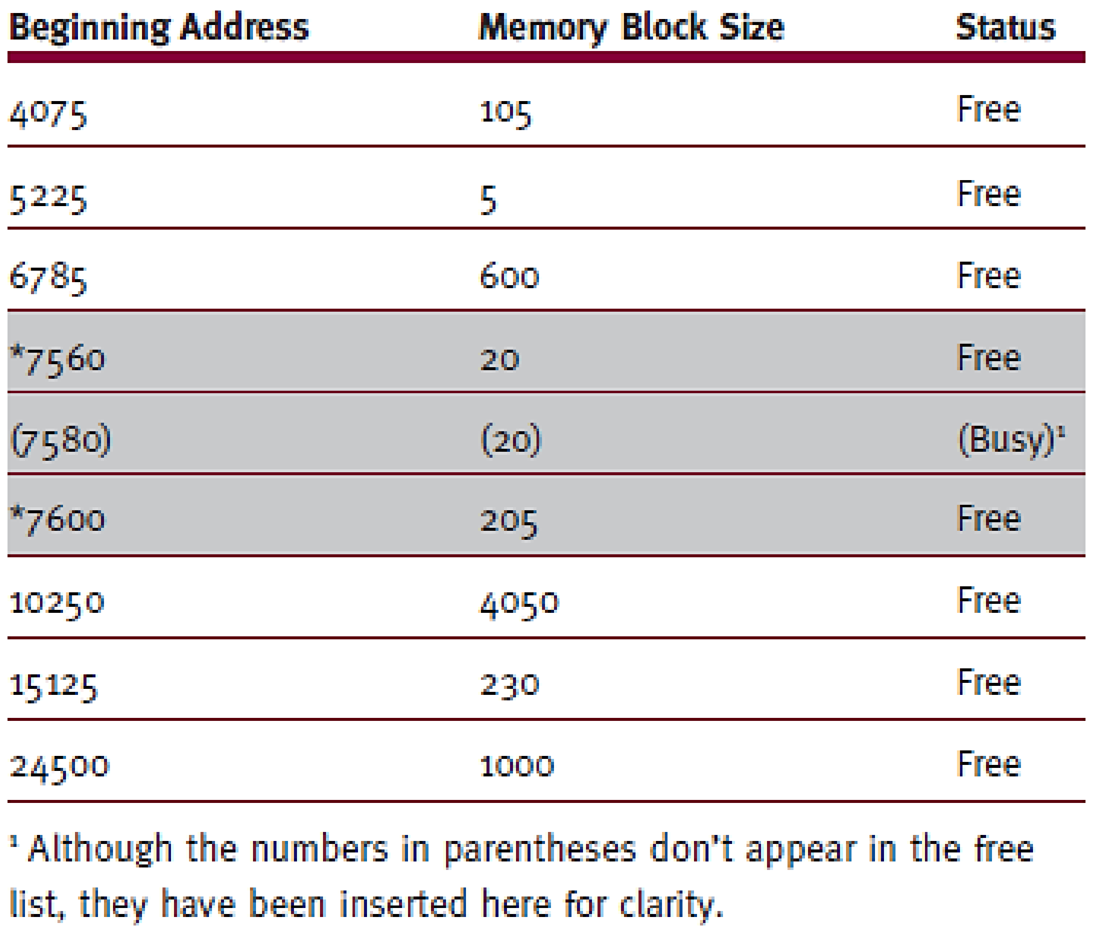
Beginning Address	Memory Block Size	Status
4075	105	Free
5225	5	Free
6785	600	Free
7560	20	Free
*7600	205	Free
10250	4050	Free
15125	230	Free
24500	1000	Free

(table 2.5)
Case 1. This is the free list after deallocation. The shading indicates the location where changes were made to indicate the free memory block (of size 205).
© *Cengage Learning 2014*

Case 2: Joining Three Free Blocks

- Deallocated memory space
  - Between two free memory blocks
- List changes: reflect starting address of new free block
  - Example: smallest beginning address (7560)
- Three free partitions’ sizes: combined
  - Example: (20 + 20 + 205)
- Total size: stored with smallest beginning address
- Last partition: assigned null entry status

Case 2: Joining Three Free Blocks (cont'd.)



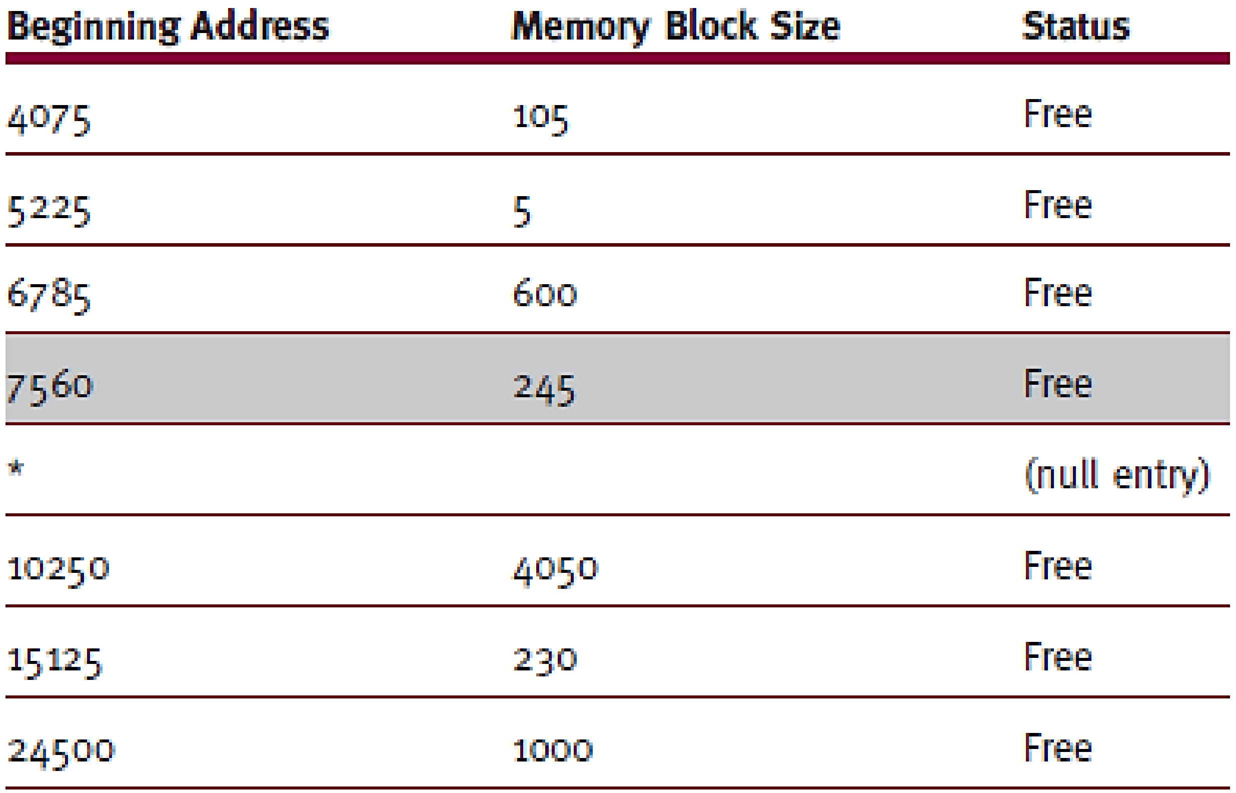
Beginning Address	Memory Block Size	Status
4075	105	Free
5225	5	Free
6785	600	Free
*7560	20	Free
(7580)	(20)	(Busy)[1]
*7600	205	Free
10250	4050	Free
15125	230	Free
24500	1000	Free

[1] Although the numbers in parentheses don't appear in the free list, they have been inserted here for clarity.

(table 2.6)
Case 2. This is the Free List before deallocation. The asterisks indicate the two free memory blocks that are adjacent to the soon-to-be-free memory block.
© Cengage Learning 2014

Case 2: Joining Three Free Blocks (cont'd.)

(table 2.7)
Case 2. The free list after a job has released memory. The revised entry is shaded.
© Cengage Learning 2014



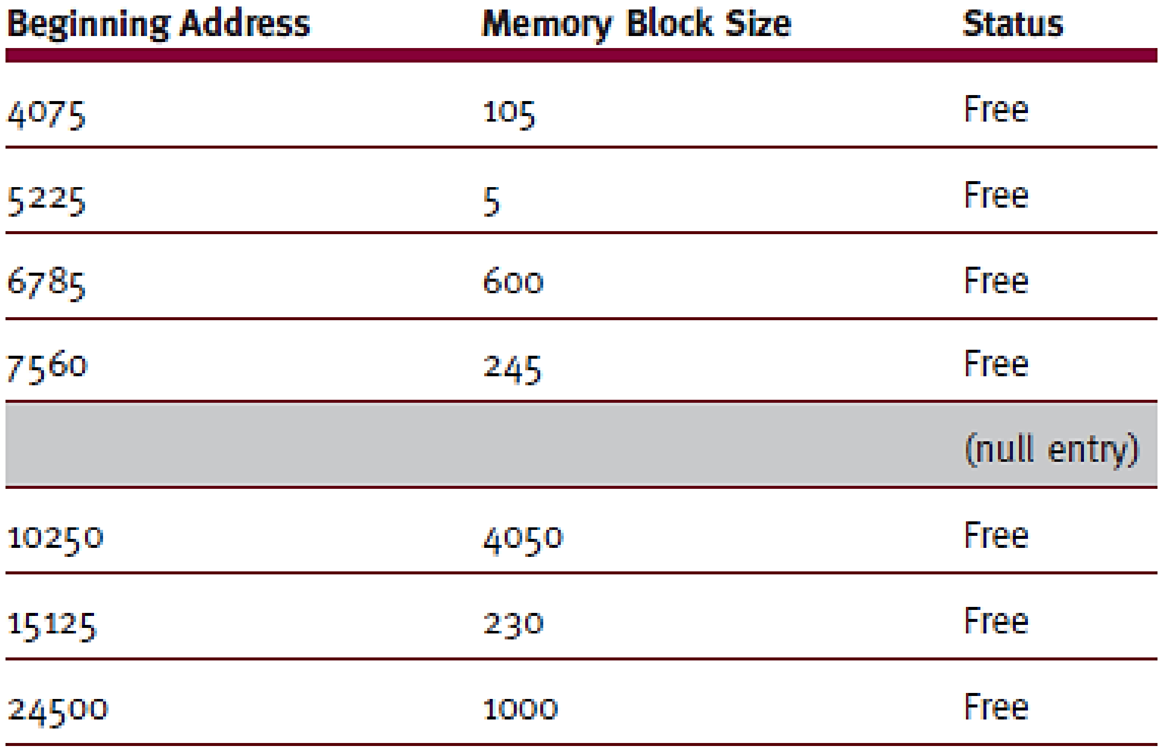
Beginning Address	Memory Block Size	Status
4075	105	Free
5225	5	Free
6785	600	Free
7560	245	Free
*		(null entry)
10250	4050	Free
15125	230	Free
24500	1000	Free

Case 3: Deallocating an Isolated Block

- Deallocated memory space
  - Isolated from other free areas
- System: determines released memory block status
  - Not adjacent to any free memory blocks
  - Between two other busy areas
- System: searches table for a null entry
  - Memory block between two other busy memory blocks: returned to the free list

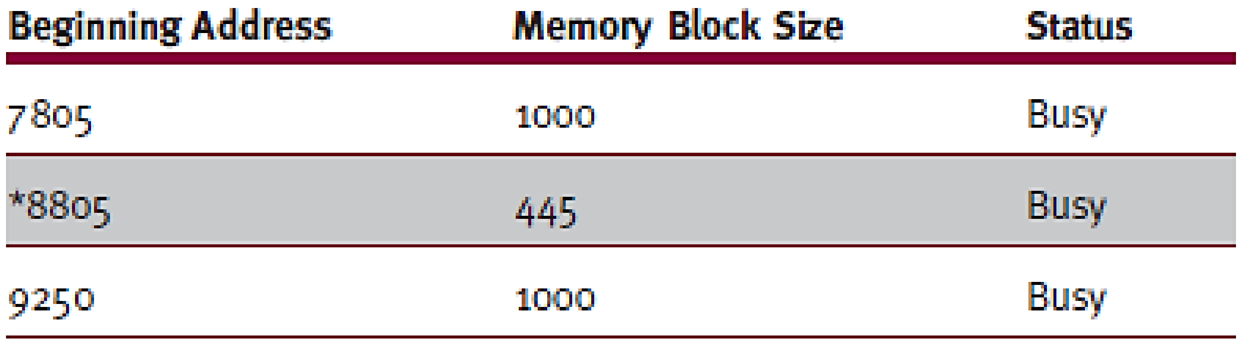
Case 3: Deallocating an Isolated Block (cont'd.)

(table 2.8)
Case 3. Original free list before deallocation. The soon-to-be-free memory block (at location 8805) is not adjacent to any blocks that are already free.
© Cengage Learning 2014



Beginning Address	Memory Block Size	Status
4075	105	Free
5225	5	Free
6785	600	Free
7560	245	Free
		(null entry)
10250	4050	Free
15125	230	Free
24500	1000	Free

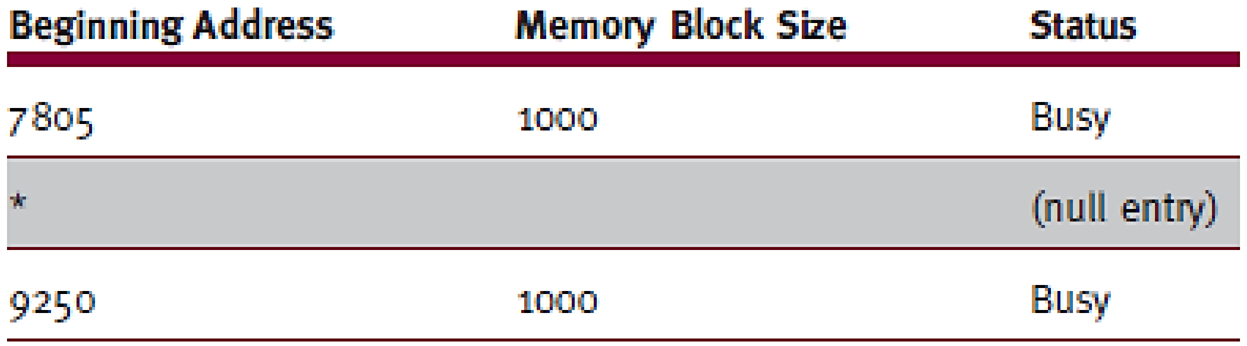
Case 3: Deallocating an Isolated Block (cont'd.)



Beginning Address	Memory Block Size	Status
7805	1000	Busy
*8805	445	Busy
9250	1000	Busy

(table 2.9)
Case 3. Busy memory list before deallocation. The job to be deallocated is of size 445 and begins at location 8805. The asterisk indicates the soon-to-be-free memory block.
© Cengage Learning 2014

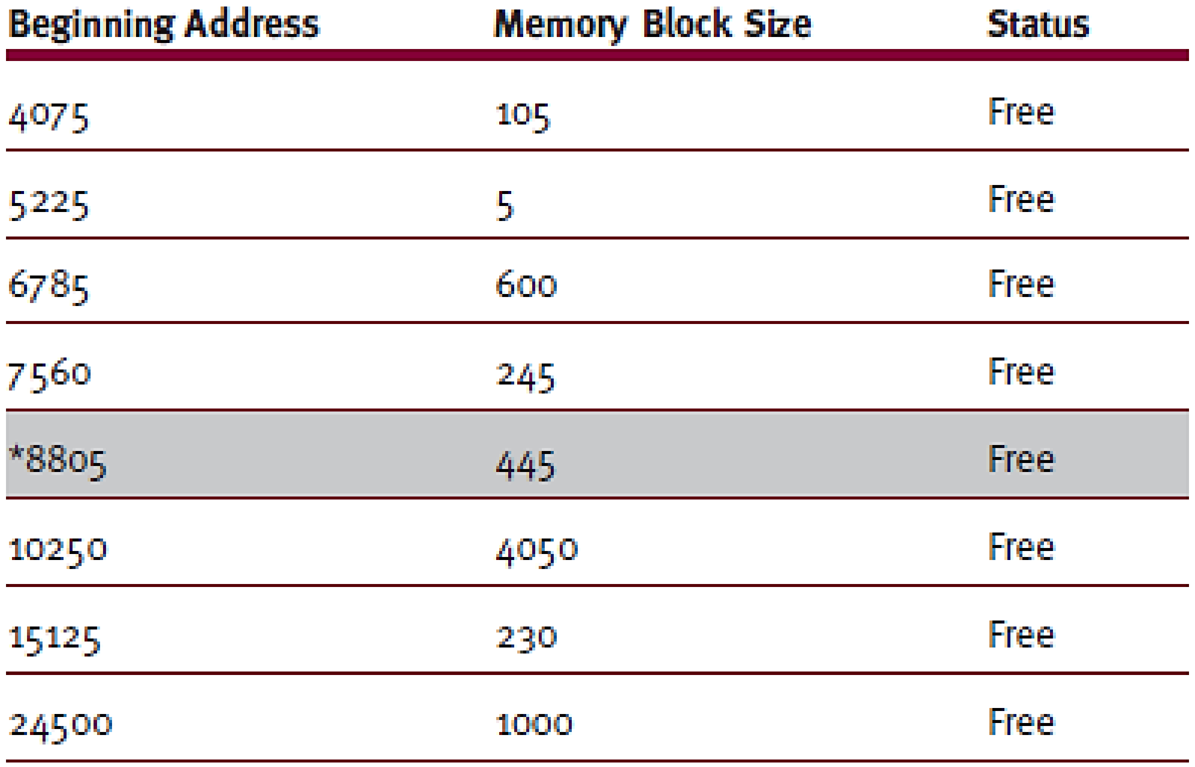
Case 3: Deallocating an Isolated Block (cont'd.)



Beginning Address	Memory Block Size	Status
7805	1000	Busy
*		(null entry)
9250	1000	Busy

(table 2.10)
Case 3. This is the busy list after the job has released its memory. The asterisk indicates the new null entry in the busy list.
© Cengage Learning 2014

Case 3: Deallocating an Isolated Block (cont'd.)



Beginning Address	Memory Block Size	Status
4075	105	Free
5225	5	Free
6785	600	Free
7560	245	Free
*8805	445	Free
10250	4050	Free
15125	230	Free
24500	1000	Free

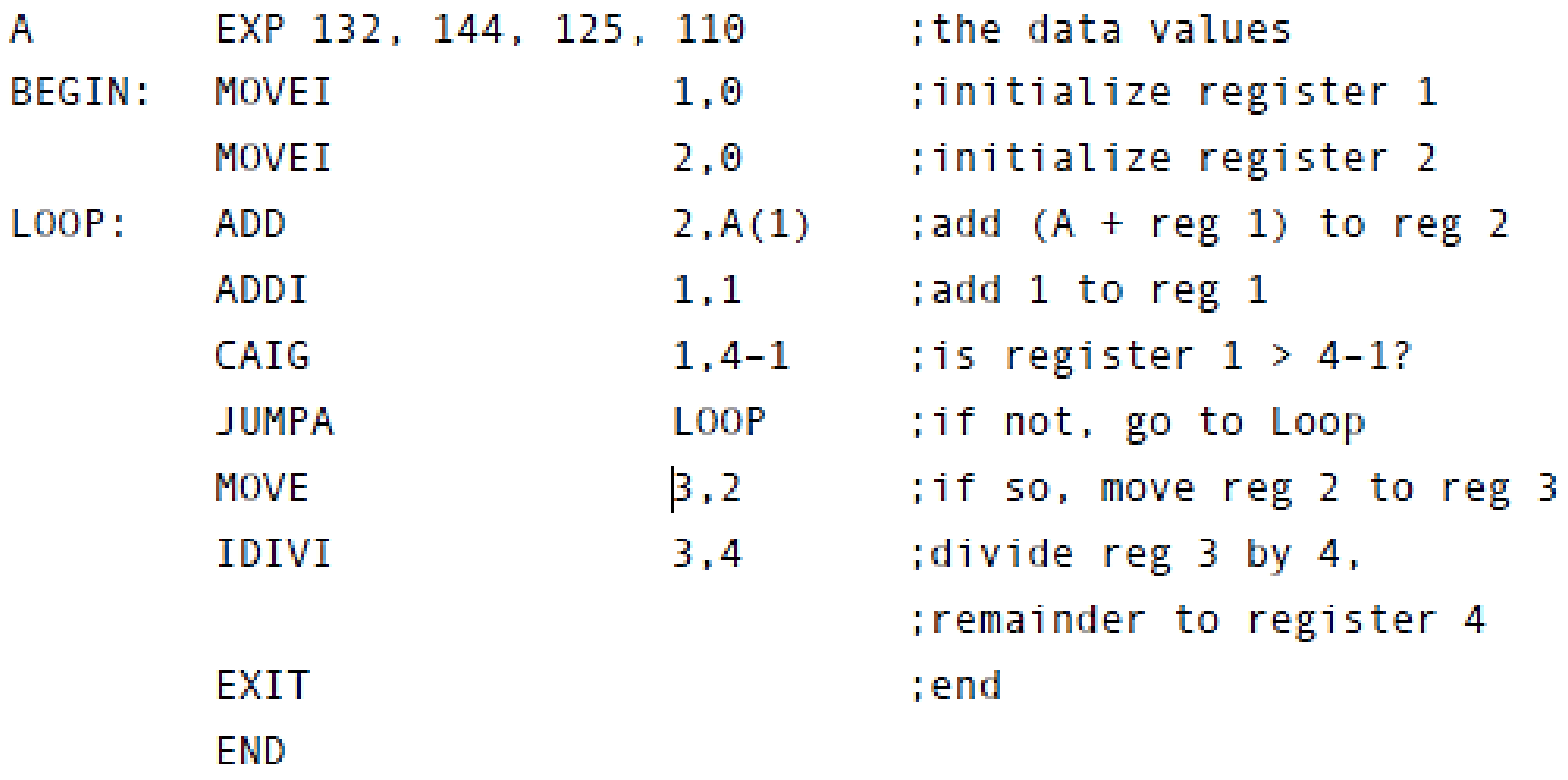
(table 2.11)
Case 3. This is the free list after the job has released its memory. The asterisk indicates the new free block entry replacing the null entry.
© Cengage Learning 2014

Relocatable Dynamic Partitions

- Memory Manager: relocates programs
  - All empty blocks: gathered together
- Empty blocks: compacted
  - Make one memory block: large enough to accommodate some or all waiting jobs

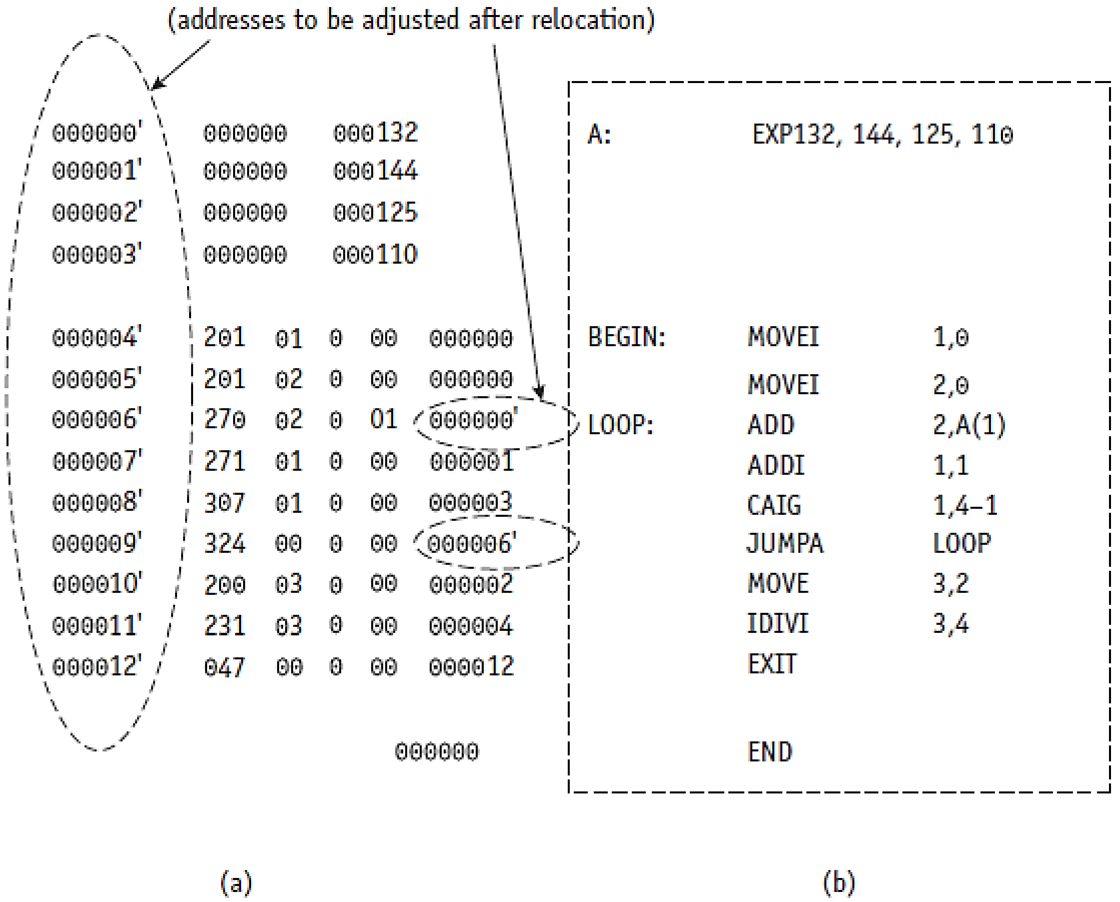
Relocatable Dynamic Partitions (cont'd.)

- Compaction (memory defragmentation): operating system reclaims fragmented memory space sections
  - Most or all programs in memory: relocated
    - Contiguous arrangement
  - Operating system: distinguishes between addresses and data values
    - Every address and address reference: adjusted to match the program’s new memory location
    - Data values: left alone



```
A         EXP 132, 144, 125, 110        ;the data values
BEGIN:    MOVEI               1,0       ;initialize register 1
          MOVEI               2,0       ;initialize register 2
LOOP:     ADD                 2,A(1)    ;add (A + reg 1) to reg 2
          ADDI                1,1       ;add 1 to reg 1
          CAIG                1,4-1     ;is register 1 > 4-1?
          JUMPA               LOOP      ;if not, go to Loop
          MOVE                3,2       ;if so, move reg 2 to reg 3
          IDIVI               3,4       ;divide reg 3 by 4,
                                        ;remainder to register 4
          EXIT                          ;end
          END
```

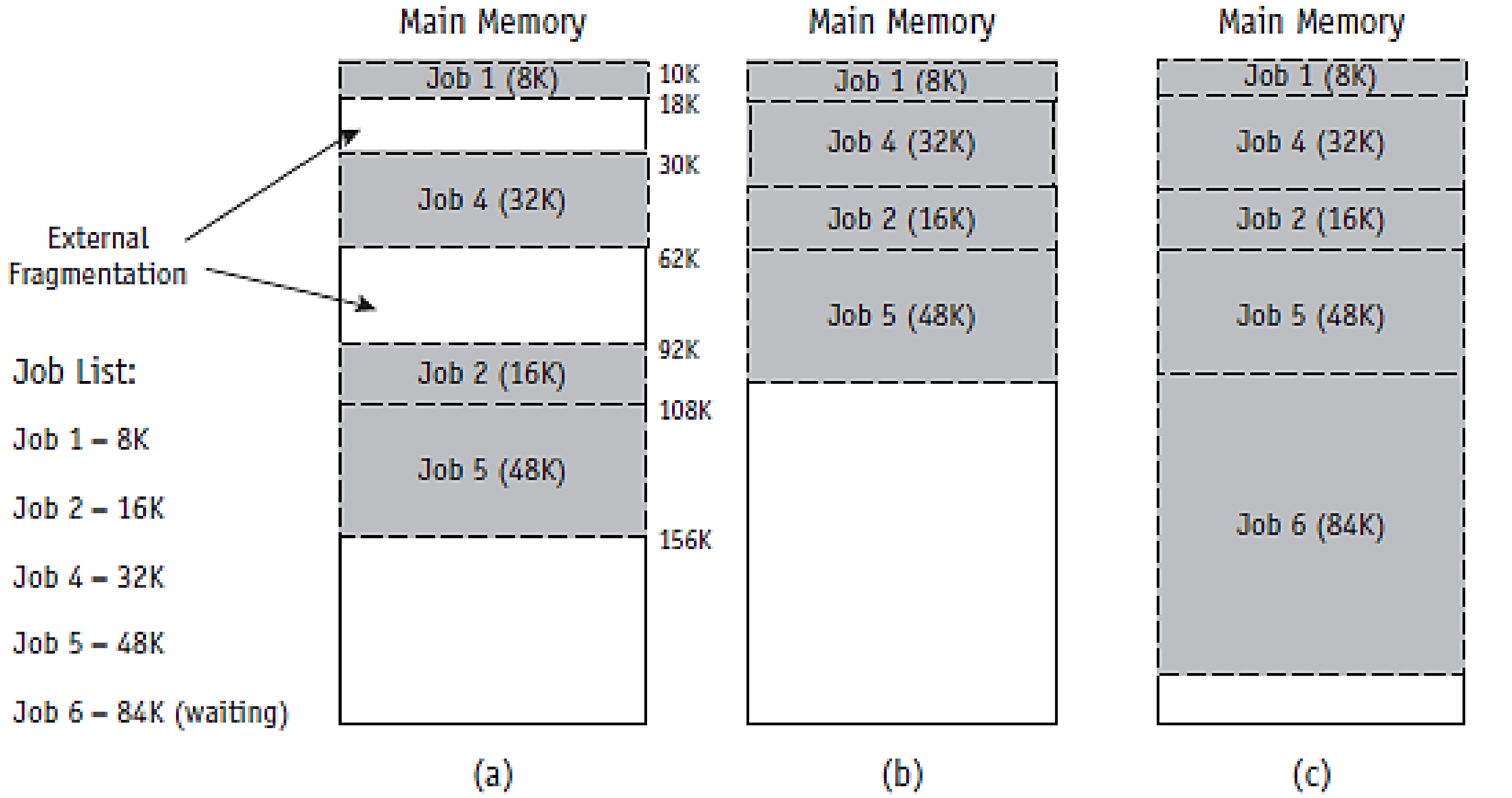
(figure 2.7)
An assembly language program that performs a simple incremental operation. This is what the programmer submits to the assembler. The commands are shown on the left and the comments explaining each command are shown on the right after the semicolons.
© *Cengage Learning 2014*



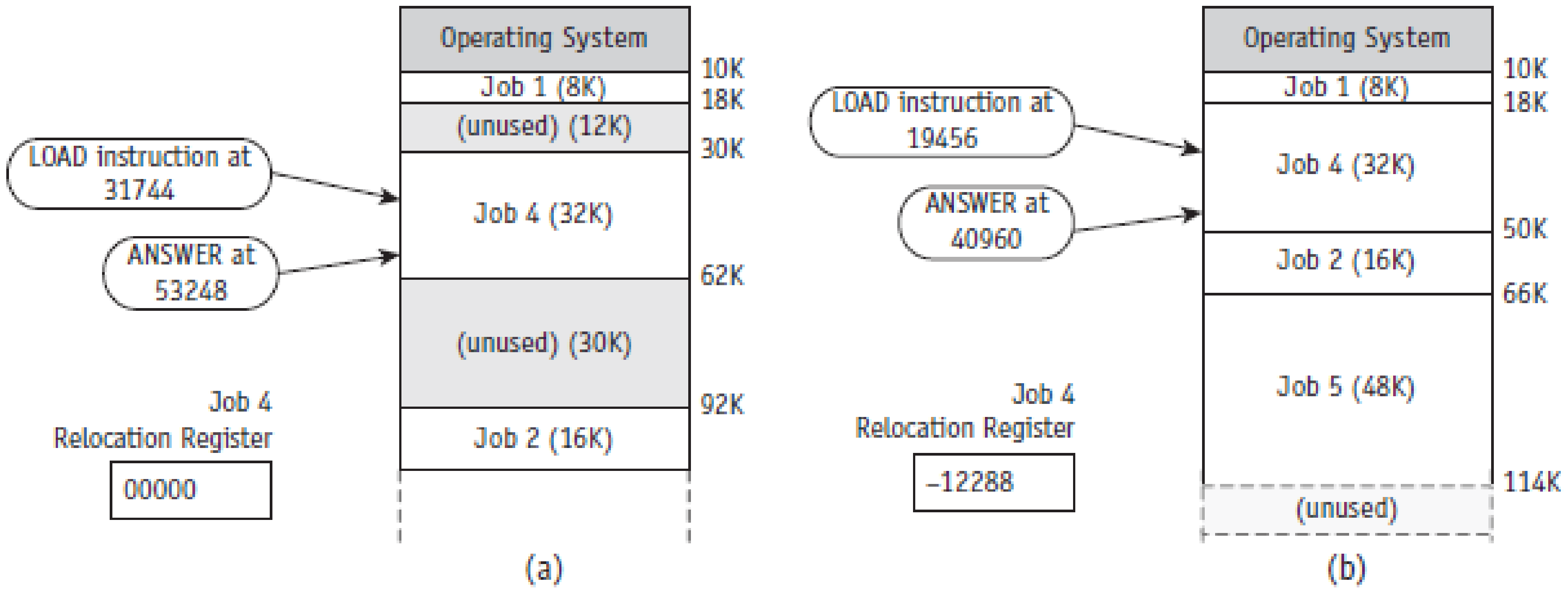
```
000000'   000000   000132
000001'   000000   000144
000002'   000000   000125
000003'   000000   000110

000004'   201  01  0  00  000000
000005'   201  02  0  00  000000
000006'   270  02  0  01  000000'
000007'   271  01  0  00  000001
000008'   307  01  0  00  000003
000009'   324  00  0  00  000006'
000010'   200  03  0  00  000002
000011'   231  03  0  00  000004
000012'   047  00  0  00  000012

                      000000
```

```
A:          EXP132, 144, 125, 110


BEGIN:      MOVEI       1,0
            MOVEI       2,0
LOOP:       ADD         2,A(1)
            ADDI        1,1
            CAIG        1,4-1
            JUMPA       LOOP
            MOVE        3,2
            IDIVI       3,4
            EXIT

            END
```

(figure 2.8)
The original assembly language program after it has been processed by the assembler, shown on the right (a). To run the program, the assembler translates it into machine readable code (b) with all addresses marked by a special symbol (shown here as an apostrophe) to distinguish addresses from data values. All addresses (and no data values) must be adjusted after relocation.
© Cengage Learning 2014



(figure 2.9)
Three snapshots of memory before and after compaction with the operating system occupying the first 10K of memory. When Job 6 arrives requiring 84K, the initial memory layout in (a) shows external fragmentation totaling 96K of space. Immediately after compaction (b), external fragmentation has been eliminated, making room for Job 6 which, after loading, is shown in (c).
© Cengage Learning 2014



(figure 2.11)
Contents of the relocation register for Job 4 before the job's relocation and compaction (a) and after (b).
© Cengage Learning 2014

Relocatable Dynamic Partitions (cont'd.)

- Compaction issues
  - What goes on behind the scenes when relocation and compaction take place?
  - What keeps track of how far each job has moved from its original storage area?
  - What lists have to be updated?

Relocatable Dynamic Partitions (cont'd.)

- What lists have to be updated?
  - Free list
    - Showing the partition for the new block of free memory
  - Busy list
    - Showing the new locations: all relocated jobs already in process
  - Each job: assigned new address
    - Exceptions: those already at the lowest memory locations

Relocatable Dynamic Partitions (cont'd.)

- Special-purpose registers: help with relocation
  - Bounds register
    - Stores highest location accessible by each program
  - Relocation register
    - Contains adjustment value: added to each address referenced in the program; “zero” value, if program not relocated

Relocatable Dynamic Partitions (cont'd.)

- Compacting and relocating: optimizes memory use
  - Improves throughput
- Compaction: overhead
- Compaction timing options
  - When a certain memory percentage is busy
  - When there are waiting jobs
  - After a prescribed time period has elapsed
- Goal: optimize processing time and memory use; keep overhead as low as possible

Conclusion

- Four memory management techniques
  - Single-user systems, fixed partitions, dynamic partitions, and relocatable dynamic partitions
- Common to all four techniques
  - Entire program: 1) loaded into memory; 2) stored contiguously; 3) resides in memory until completed
- All schemes
  - Place severe restrictions on job size: limited by the largest petition
  - Groundwork for more complex memory management